

**SUBMICRON SYSTEMS ARCHITECTURE PROJECT**

Department of Computer Science  
California Institute of Technology  
Pasadena, CA 91125

**Semiannual Technical Report**

Caltech Computer Science Technical Report

**Caltech-CS-TR-89-12**

31 October 1989

The research described in this report was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202; and monitored by the Office of Naval Research under contract number N00014-87-K-0745.

# **SUBMICRON SYSTEMS ARCHITECTURE**

## **Semiannual Technical Report**

*Department of Computer Science  
California Institute of Technology*

**Caltech-CS-TR-89-12**

**31 October 1989**

**Reporting Period: 1 April 1989 – 31 October 1989**

**Principal Investigator: Charles L. Seitz**

**Faculty Investigators: K. Mani Chandy  
Alain J. Martin  
Charles L. Seitz  
Stephen Taylor**

**Sponsored by the  
Defense Advanced Research Projects Agency  
DARPA Order Number 6202**

**Monitored by the  
Office of Naval Research  
Contract Number N00014-87-K-0745**

# SUBMICRON SYSTEMS ARCHITECTURE

*Department of Computer Science  
California Institute of Technology*

## 1. Overview and Summary

### *1.1 Scope of this Report*

This document is a summary of research activities and results for the seven-month period, 1 April 1989 to 31 October 1989, under the Defense Advanced Research Project Agency (DARPA) Submicron Systems Architecture Project. Previous semiannual technical reports and other technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

### *1.2 Objectives*

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes. Our work is focused on VLSI architecture experiments that involve the design, construction, programming, and use of experimental message-passing concurrent computers, and includes related efforts in concurrent computation and VLSI design.

### *1.3 Highlights*

- Memoryless Mosaic functional on first silicon (sections 2.1 and 4.9).
- 192-node Symult Series 2010 multicomputer (section 2.2)
- Program Composition (section 3.1)
- Cantor for the Mosaic (section 3.2)
- Testing the asynchronous microprocessor (section 4.1).
- The limits of delay-insensitivity (section 4.2).
- Self-timed mesh-routing chips operate at 65MB/s (section 4.7).



## 2. Architecture Experiments

### 2.1 Mosaic Project

*Chuck Seitz, Nanette J. Boden, Jakov Seizovic, Don Speck, Wen-King Su, Steve Taylor, Tony Wittry*

The Mosaic C is an experimental fine-grain multicomputer, currently in development. Each Mosaic node is a single VLSI chip containing a 16-bit processor, a three-dimensional mesh router, a packet interface, 16KB of RAM, and a ROM that holds self-test and bootstrap code. These nodes are arrayed logically and physically in a three-dimensional mesh. We are working toward building a 16K-node ( $32 \times 32 \times 16$ ) Mosaic prototype, together with the system software and programming tools required to develop application programs.

The Mosaic can be programmed using the same reactive-process model that is used for the medium-grain multicomputers that our group has developed. However, the small memory in each node dictates that programs be formulated with concurrent processes that are quite small. The Cantor programming system supports this style of reactive-process programming by a combination of language, compiler, and runtime support. The programmer is responsible only for expressing the computing problem as a concurrent program. The resources of the target concurrent machine are managed entirely by the programming system.

The Mosaic project includes many subtasks, which are listed below together with their current status:

**Design, layout, and verification of the single-chip Mosaic node.** The Mosaic C chip with 16KB of memory is  $9.0\text{mm} \times 7.4\text{mm}$  in a  $1.2\mu\text{m}$  CMOS process, and has 84 pads. Yield characterization indicates that a node with 16KB rather than 8KB of primary memory will increase the chip fabrication cost by less than 30%. Doubling the primary memory at  $1.3 \times$  the cost for the prototype is a good tradeoff. Additional memory will be particularly helpful for a system that will be used extensively for software development. A substantial economy has been achieved by using TAB rather than conventional packages, so the total fabrication budget has not changed from original estimates.

A "memoryless Mosaic" test chip containing the processor, packet interface, router, clock driver, and central timing and memory arbitration was sent to MOSIS on August 10th to be fabricated in the  $1.6\mu\text{m}$  SCMOS process. (The memory section had been verified earlier.) These chips were returned from fabrication on October 12th, and have been subjected to preliminary tests. Although there are additional tests to perform, this chip appears to operate completely correctly on first silicon, with a yield of 47/50 in the preliminary screening. All processor instructions and the router have been tested; the packet interface is now being tested. The test fixture currently limits speed testing to a clock period of 37ns (27MHz). The chip operates correctly with a clock period of 37ns, except for one case. When an incoming packet

directs the router to switch the packet onto the next dimension, the minimum clock period for correct operation is approximately 65ns. Depending on the nature of the design error, this problem may require a design iteration on the memoryless Mosaic. (See section 4.9 for additional details.)

**Internal self-test and bootstrap code.** Since the Mosaic C is a programmable computing element, devoting a portion of the bootstrap ROM to self-testing greatly simplifies the logistics of producing these chips in quantity. The bootstrap and self-test code will be tested with EPROM connected to the memoryless Mosaic elements. Additional tests to the channels, which must be accomplished by the fabricator's automatic test equipment, are being written.

**Packaging.** The packaging design is based on Tape Automated Bonding (TAB) of the chips on small circuit boards. The manufacturing and replacement unit will contain eight nodes in a logical  $2 \times 2 \times 2$  submesh. These modules have stacking connectors that provide 160 pins on both the top and bottom, and are confined by pressure between motherboards to provide a three-dimensional connection structure that can be disassembled and reassembled for repair. We are currently evaluating suitable connectors.

**Cantor runtime system.** A Cantor runtime system has been written in Mosaic assembly code, and is now interfaced to the code produced by version 3.0 of the Cantor programming system. Research is underway on runtime algorithms that allow the system to operate robustly in spite of fluctuations in local storage demands. For example, if a local receive queue threatens to overflow, a part of the receive queue is distributed to another node. (See also section 3.2.)

**Cantor language, compiler, and application studies.** We are now experimenting with version 3.0 of the Cantor language and compiler, which was developed by William C. Athas at the University of Texas at Austin.

**Host interfaces and displays.** The three-dimensional mesh structure of the Mosaic allows a very large bandwidth around the mesh edges. In order to initiate and interact with computations within the Mosaic, we are designing interfaces between the Mosaic message network and host computers, and between the message network and displays.

A system that will serve both as a prototype of a host interface and as a software development platform is based on eight memoryless Mosaic elements connected to fast, two-ported, external memories. This workstation add-in board will provide an interface that will allow the workstation to monitor the memories of the Mosaic elements during program execution.

In order to provide a high-performance display capability for the Mosaic, we have designed a system that uses one  $32 \times 32$  plane of a Mosaic as a rendering engine and frame buffer. A detailed design of the video output generator that attaches to one edge of this  $32 \times 32$  plane has been completed; construction awaits finalization of packaging decisions.



## 2.2 Second-Generation Medium-Grain Multicomputers\*

*Chuck Seitz, Joe Beckenbach, Christopher Lee, Jakov Seizovic, Craig Steele, Wen-King Su*

Symult Systems has delivered additional contributed equipment over the past seven months, with the result that we are now operating a 192-node Symult Series 2010 multicomputer for applications and a 32-node Symult Series 2010 for system development. Utilization of the 192-node system through the Caltech Concurrent Supercomputer Facilities has been at a level of approximately 88% of the available node-hours. These systems run very dependably, and have yet to exhibit a hardware failure.

Copies of the Cosmic Environment system have been distributed on request to 20 additional sites during this period, bringing the total copies distributed directly from the project to nearly 200.

We are implementing a new version of the Cosmic Environment host runtime system, and adding numerous new features to the Reactive Kernel node operating system. The new CE is based internally on reactive-process programming, and will allow a more distributed management of a set of network-connected multicomputers. The extended RK will support global operations across sets of cohort processes, including barrier synchronization, sum, min, max, parallel prefix, and rank. Another extension will be the support of distributed data structures, such as sets and ordered sets. These new features will be implemented at the RK handler level, where the message latency is only a fraction of that at the protected user level. The implementation of these algorithms at the handler level permits global and distributed-data-structure operations in times that do not greatly exceed those of user-level operations dealing with single messages.

Our Caltech project continues to work closely with DARPA-supported Touchstone project at Intel Scientific Computers. Our contributions include the architectural design, message-routing methods and chips, and system software. (See section 3.3 for a summary of the port of RK to the iPSC/2, and section 4.7 for a summary of test results on mesh-routing chips.)

The Cosmic Cubes that were built in our project in 1983 continue to operate reliably. No hard failures were recorded in this seven-month period. The two original Cosmic Cubes have now logged 4.2 million node-hours with only four hard failures; three of these were chip failures in nodes, and one a power-supply failure. A node MTBF in excess of 1,000,000 hours is probable based on this reliability experience.

---

\* This segment of our research is sponsored jointly by DARPA and by grants from Intel Scientific Computers (Beaverton, Oregon) and Symult Systems (Monrovia, California).

### 3. Concurrent Computation

#### 3.1 Program Composition

*K. Mani Chandy, Steve Taylor*

This research investigates the use of program composition as a method of developing concurrent programs. The goal is to develop a theory, a notation, and an implementation of program composition operators so that programs can be developed by putting smaller programs together to get larger ones. The compositional approach to programming was described in the previous semiannual technical report. New components of this work are:

1. A primitive set of composition operators (and not merely sequential or functional composition) has been implemented, and a proof theory has been developed for this set of operators.
2. The researchers believe that in each application area there are a few problem-solving paradigms or “templates,” and that, formally, these templates are user-defined composition operators. Thus, the notation allows user-defined composition operators.
3. The notation is intended to execute on both shared-memory and message-passing concurrent computers, without modification. A fragment of the notation has been implemented on the Connection Machine by Professor Rajive Bagrodia at UCLA.
4. The theory incorporates functional programming ideas, and extends it to problems that are not functional. (Most reactive systems are nondeterministic, and nonfunctional.)
5. The researchers have been working with computational fluid dynamicists and biologists to identify problem-solving paradigms in these disciplines, and to evaluate whether the compositional approach is effective in these areas.

The theory of program composition has been developed, and a prototype implementation in Strand has been completed. Discussions with Caltech faculty in Applied Math and Biology have provided initial test cases. Discussions with researchers at Aerospace Corporation have allowed an evaluation of program composition for tracking and trajectory-computation applications, and have led to initial joint research in these applications.

#### 3.2 Cantor for the Mosaic

*Nanette J. Boden, Chuck Seitz*

With the Cantor version 3.0 compiler and interpreter in place, we are beginning to translate a representative subset of our library of Cantor application programs into the new version. The purpose of this exercise is twofold: We maintain a library of programs for demonstrations, and we continue the process of evaluating the impact

of new language features on application programming. The aspects of the Cantor 3.0 that have the most impact on programming are the incorporation of functions and the introduction of message discretion.

As usual in the development of programming systems, the introduction of new capabilities at one level of the system imposes new requirements at other levels. In the case of the new features of Cantor 3.0, the introduction of message discretion raises the specter of violating the guarantee of message consumption. If a process is waiting for the arrival of a particular message, messages received in the interim must be buffered. Since the resources *of a node* are quite limited, physical space may not be available for the awaited message to be received. Since infinite queueing is theoretically required, we are investigating engineering solutions that use the resources *of the entire machine*, and potentially of secondary memory, to approximate infinite queues.

In addition to implementing runtime support for new language features, we are investigating solutions to other problems that became apparent during the development of the Mosaic runtime system. In this first version, we made simplifying assumptions to minimize both the size and complexity of the runtime support. Two of the assumptions that must be seriously addressed in future versions of the runtime system are: (1) if an available reference value exists for the creation of a new process on a remote node, then enough resources exist on that node for the new process; and (2) the code for each process resides on every node. These assumptions are clearly unrealistic for the types of memory-intensive computations that we seek to perform. Currently, we are devising and evaluating schemes for process placement that do not assume available resources on the remote node. We are also devising schemes for code partitioning that will maximize the amount of memory available for processes, while not introducing excessive overhead for acquiring necessary copies of process code.

### **3.3 The Cosmic Environment and Reactive Kernel**

*Chuck Seitz, Joe Beckenbach, Christopher Lee, Jakov Seizovic, Wen-King Su*

A joint effort with Intel to port the Reactive Kernel to run as the native node operating system on the iPSC/2 has successfully achieved its first milestone. Our longer-term goal is to run an enhanced version of RK on a future Intel multicomputer that is based on the Intel i860 processor.

The port of the Inner Kernel of RK and of the system-handler layer was performed in an intensive effort over a two-week period by Jakov Seizovic, RK's original author, and was upgraded to include a preliminary user-process handler by Bill Bain of Intel during the following two weeks. The fine-tuning of the message performance took another week. This port has shown once again that the modular structure of RK provides for simple porting and simplifies debugging, especially in the early phases of the port. This preliminary version of RK outperformed



the Intel NX operating system by about a factor of two in message latency, and achieved equivalent message bandwidth. We have subsequently increased the message bandwidth while providing proper fragmentation and reassembly of long messages, which increases the fairness of access to the message network. The completion of this port is expected to be performed principally by Intel within the next two months.

RK has gotten somewhat ahead of the Cosmic Environment system in its use of a layered reactive-process structure. A new version of CE has been designed, and is currently being written.

### 3.4 Hybrid Distributed Discrete-Event Simulators

*Wen-King Su, Chuck Seitz*

Two hybrid distributed simulators have been written, and their performance results are included in the PhD thesis: "Reactive-Process Programming and Distributed Discrete-Event Simulation," [Caltech-CS-TR-89-11].

In a distributed discrete-event simulation, the simulation subject is divided into a number of smaller elements. The elements are distributed over a multicomputer or a multiprocessor, and are simulated concurrently. In a conservative simulator, null messages are necessary for the progress of a circuit of idling elements. In the framework of the Chandy-Misra-Bryant algorithm, elements are simulated independently, as if each element is located on a separate node. While this framework will achieve good performance on a fine-grain multicomputer, the volume of null messages is an unnecessary burden for a medium-grain multicomputer, in which many elements share the same node. When nodes are few, the CMB simulator does worse than a sequential simulator.

The goal of the hybrid simulators is to eliminate intra-node null messages by combining elements on the same node into a single macro-element. In the hybrid-1 simulator, macro-elements are simulated internally by a conventional sequential simulator. Hybrid-1 reduces intra-node messages by eliminating all intra-node null messages. It also reduces inter-node messages by synchronizing all element outputs in a macro-element. The result is a simulator that equals a sequential simulator on a single node and shows a speedup when more nodes are used, regardless of element placement. However, the amount of speedup is limited because some concurrency is lost to the strict synchronization. In hybrid-2, macro-elements are simulated by a combination of CMB and sequential simulators. Elements are constantly moved between the two modes as they become blocked or unblocked. Since an element can progress as far as its inputs allow, the hybrid-2 can attain the full CMB speedup when many nodes are used. However, since element outputs are not synchronized in each macro-element, hybrid-2 is sensitive to element placement.

### 3.5 CONCISE\*

*Sven Mattisson, Lena Peterson, Chuck Seitz*

The concurrent circuit-simulation program, CONCISE, originally used waveform relaxation in conjunction with Jacobi iterations. This method gives high concurrency, but other iterative methods have better convergence performance. These other methods do not, however, offer the same concurrency as the Jacobi method. Thus, we have concentrated recently on developing combinational methods that retain the concurrency properties of the Jacobi iterations while improving convergence. CONCISE has been enhanced to exploit circuit-node coupling. The strongly coupled nodes are solved in a block with a direct method; thus, convergence is improved.

The waveform relaxation method has also been augmented with multicolored Gauss-Seidel iterations. Normally, Gauss-Seidel iterations rely on the equations being solved in sequence. However, by coloring the circuit graph it is possible to find an ordering that gives high concurrency. All equations with one color can be solved in parallel, and typically only three to five colors are needed for a circuit to yield high concurrency.

A special version of CONCISE was written to evaluate Jacobian matrix coefficients concurrently, while using a single-rate integration method for each subsystem. This version is now about to be incorporated in the standard version.

A plotting program, communicating with CONCISE via messages, has been developed. This program displays selected waveforms as they are computed.

CONCISE was given a thorough workout over the summer performing simulations on a 64-node Symult 2010 of 4000-transistor sections of the FMRC2.1 self-timed mesh-routing chips. These studies were part of characterizing the process-dependence of the FMRC2.1 design.

CONCISE is written in C using the CE/RK functions, and now runs on Sun, Sequent, Macintosh II (A/UX), Intel iPSC/1, Intel iPSC/2, and Symult Series 2010 computers.

### 3.6 A C-Based Concurrent Programming Language For Multicomputers

*Marcel van der Goot, Alain Martin*

We are defining and implementing a concurrent programming language for message-passing multicomputers. Since the main difference between multicomputers and sequential machines is the possibility of concurrency, we have concentrated in our language design on adding concurrency without redefining the complete computation model. In particular, since most of our programming experience is

---

\* This segment of our research is a joint project with the Applied Electronics Department of the University of Lund, Sweden.

with using imperative sequential languages, we have chosen one such language, C, as the basis for our work. C matches well with our desire to design a language that is compact but nevertheless useful for writing “real” application programs.

In our model, a computation consists of a set of independently executing sequential processes, plus a set of message-buffers (channels) connecting pairs of processes. Processes and channels together form the so-called computation graph, which can vary dynamically during the computation. A process is a short sequential (C) program that can exchange data with its environment by sending or receiving messages. A process typically has about the same size as a function; such a fine grain size makes the language applicable to a large range of multicomputers.

We finished a preliminary implementation of a somewhat restricted version of the language earlier this summer. In that implementation, a concurrent program is compiled into a single UNIX process that is executed on a Sun workstation. Currently we are working on a compiler for the complete language, which we hope to have running in December or January.

## 4. VLSI Design

### 4.1 Testing of the Asynchronous Microprocessor

*Steve Burns, Tony Lee, Dražen Borković, Pieter Hazewindus, Alain Martin*

The Asynchronous Microprocessor, described in the previous semiannual technical report, has since been thoroughly tested. Chips fabricated at a  $2\mu\text{m}$  feature size functioned as intended over a wide range of power supply voltages, temperatures, and delays of the external memories. The chips fabricated at  $1.6\mu\text{m}$ , while functioning correctly at certain voltages, temperatures, and delays, failed for many values of these external parameters. After a detailed analysis, we concluded that all the high-level transformations were performed correctly. The problem, instead, occurred in the final phase of the compilation, the transformation from production rules into networks of CMOS gates. In particular, the values of some isochronic forks change too slowly, allowing different gates to interpret the digital value inconsistently. These forks were located and the circuits were modified to correct the problem. A corrected  $1.6\mu\text{m}$  version of the microprocessor is expected back from MOSIS fabrication on December 1st.

### 4.2 The Limitations to Delay-Insensitivity in Asynchronous Circuits

*Alain Martin*

Once it was established that the problem in the  $1.6\mu\text{m}$  version of the microprocessor was caused by a malfunctioning of an isochronic fork for certain values of the external parameters, the question of whether isochronic forks are necessary needed to be answered.

An isochronic fork is used to distribute a variable to several points of the circuit as input of several gates. In the discrete model, it is assumed that the different copies of the variable have the same values at all times. For this assumption to be valid, the following timing requirement has to be fulfilled. A change on the input of a fork causes the different outputs to change asynchronously. However, the "transition delays" on the different outputs of an isochronic fork must be similar enough in length that once a change on one of the outputs of the fork has caused another gate to fire, one may conclude that the changes on all the outputs have completed.

Since the definition of isochronic forks violates the delay-insensitivity assumption, and since all efforts to design entirely delay-insensitive circuits have been fruitless, we started to suspect that the class of circuits that are entirely delay-insensitive could be very limited. Indeed, we have been able to prove that an entirely delay-insensitive circuit can contain only C-elements, hence settling an important open question in the theory of asynchronous circuit design, and vindicating the compromise to delay-insensitivity implied by the use of isochronic forks.



### 4.3 Tools for Performance Evaluation of Self-timed Circuits

*Steve Burns, Alain Martin*

The compilation method has, in the past, been mostly concerned with correctness, not efficiency. With the design of the microprocessor, high performance has become a major concern. Two separate analysis tools have been developed in order to determine the speed at which self-timed circuits operate.

The first tool is a simple event-driven simulator that takes, as input, extracted circuit layout. Timing analysis is based on the  $\tau$ -model. Good agreement has been found between the timing information produced by the simulator and actual results obtained from the fabricated chips. The simulator itself is quite efficient, even for large circuits; simulation of a single instruction of the microprocessor takes less than a second.

The second tool allows the comparison of various methods of handshaking without actually constructing and then simulating the circuit. The fundamental sequencing between actions can be determined, in many important cases, by a static analysis of a high-level description of the program. The necessary analysis involves solution of a finite linear optimization problem. For small problems, it can be solved by the enumeration of all the cycles in a so-called "constraint" graph. A PROLOG program has been constructed that performs this analysis.

### 4.4 Cache Memory for an Asynchronous Microprocessor

*José A. Tierno, Alain Martin*

The design of a direct-mapped instruction cache for an asynchronous microprocessor is underway. The cache is completely self-timed, both the control part and the RAM array. The objective is to make the design suitable for on-chip implementation as part of the processor pipeline.

### 4.5 Self-Timed Circuits in GaAs

*José A. Tierno, Alain Martin*

Experimentation is being done on new transistor configurations for digital circuits implemented in Enhancement/Depletion mode MESFET GaAs technology. The main characteristics of these configurations are increased noise margins, reduced input load, and slightly faster gate delays than conventional DCFL (direct-coupled FET logic) and SBFL (super buffered fet logic) technology. Extensive experimentation has been done using SPICE for simulations, and two chips have been sent for fabrication to test some basic circuits.

## 4.6 Testing Self-Timed Circuits

*Pieter Hazewindus, Alain Martin*

We are continuing our investigation into the testability of self-timed circuits. Previously we tried to construct a set of circuit elements with which any program could be implemented and for which all faults would be testable. This goal seems unattainable: We have found that – with one exception – for any isochronic fork there is a corresponding fault that is not testable. As we have shown that most circuits require isochronic forks, the range of circuits without untestable faults is very limited. Hence, without additional circuitry or additional scan points, most circuits will have untestable faults.

To increase the fault coverage it is possible to add a test structure, thus connecting all state-holding elements in a queue and thereby reducing the problem of testing a sequential circuit to that of testing a combinational one. Test vectors are put into the queue, while the results are taken out, similar to scan-type designs for synchronous circuits. We speculate that with appropriate conditions on the combinational logic, all faults are testable this way; however, for our current design style, such a queue would be expensive in area, as the number of state-holding elements is much larger than the number of latches in a typical synchronous design. We are investigating ways to reduce the number of state-holding elements in the queue while maintaining the complete testability.

## 4.7 Fast Self-Timed Mesh Routing Chips

*Chuck Seitz*

The FMRC2.1 mesh-routing chips have now been thoroughly characterized by Intel, and have been shown to operate at a channel rate of 65MB/s. However, testing at Intel also discovered a failure mode that occurs when several channels operate concurrently. This failure was traced to collapse of the internal power supply under these demanding conditions; thus, it is properly a failure of the packaging rather than of the chip design.

This 132-pin chip devotes the 20 lowest-inductance PGA-package pins to Vdd and GND, but either a better package or twice as many Vdd and GND pins are required. Experiments with a number of alternative packages are now underway, and have involved producing a complete set of test vectors for automatically testing MRC chips.

The design and layout of two other versions of the FMRC is now underway. One of these versions is designed to minimize latency by using relatively few internal FIFO stages. Multicomputer applications benefit from the internal FIFOs, which reduce blocking contention, but the tradeoff between throughput and latency is different for multiprocessor applications.

Samples of tested FMRC2.1 chips have been provided in recent months to CMU,



Analytical and simulation results for this situation were nearly identical. For simulation simplicity, the set of three, parallel, minimum-width wires in each vertical bundle was treated as one wide wire. Only the left half of the distribution network was simulated with SPICE. The result of these simulations confirmed the following area-energy-period optimums: (1) a clock driver of 700sq  $n$ -channel + 1050sq  $p$ -channel *per half* of the RAM per phase, and (2)  $15\lambda$ -wide distribution wires across the top for each phase.

The memory and router sections of the Mosaic were fabricated and tested more than a year ago. To test the remaining parts of the full Mosaic element and their ability to work together, the processor, packet interface, router, and clock driver were integrated onto a "memoryless Mosaic" test chip that was sent to MOSIS on August 10th. This test chip was a major milestone for us. A number of improvements in the processor instruction set and an increase in the channel bandwidth created some design imbalances that required a substantial amount of rethinking of the memory arbitration and packet interface.

The maximum combined data bandwidth of the receive and send parts of the packet interface (PI) is equal to 50% of the total memory bandwidth, which is one 16-bit read or write each clock period (25ns). The original design specifications for the PI included the assumption that there would be enough spare memory cycles so that the PI would not need to request access to the data bus; it would instead use otherwise unused cycles for message transfer from/into the network.

The increased efficiency of the processor microcode invalidated this assumption, and the design of the memory-bus arbitration unit and the PI had to be modified to comply with the new specifications. Eight-word buffers were added between the memory and the sending part of the packet interface, and between the receiving part of the packet interface and the memory. The signals generated by the sender and the receiver part of PI to request access to the memory bus include "hysteresis": Depending on the amount of space available in the buffers, the PI may either steal unused cycles or request exclusive use of the memory. This scheme allows the data transfer between memory and buffers to occur in bursts, rather than imposing the bus arbitration overhead on every PI memory access.

The new design provides for the data transfer from/into the network at the full network bandwidth regardless of the instruction sequence being executed. This feature was achieved with a fairly modest increase in complexity: an increase in buffering space from two to sixteen words, and an additional state machine to handle the bus-request logic.

We are currently in the process of testing the memoryless Mosaic chips that were returned from MOSIS fabrication on October 12th. So far, the chips appear to function correctly. All processor instructions and router functions operate correctly, but there is evidently a slow path in the router. We are investigating this problem.



# The Limitations to Delay-Insensitivity in Asynchronous Circuits

Alain J. Martin  
Department of Computer Science  
California Institute of Technology  
Pasadena CA 91125, USA

Asynchronous techniques—i.e. techniques that do not use clocks to implement sequencing—are currently attracting considerable interest for digital VLSI circuit design, in particular when the circuits produced are *delay-insensitive*. A digital circuit is delay-insensitive (d.i.) when its correct operation is independent of the delays in operators and in the wires connecting the operators, except that the delays are finite and non-negative.

In this paper, we characterize the class of circuits that are entirely delay-insensitive, and we show that this class is surprisingly limited: Practically all circuits of interest fall outside the class since circuits inside the class may contain only C-elements as multi-input operators.

## 1. Circuits as Networks of Gates

A d.i. circuit is a network of logical operators, or *gates*. A gate has one or more Boolean inputs and one Boolean output. A gate represents a Boolean function: For constant values of the inputs, the output takes a value that is defined by a Boolean function of the inputs, and possibly the current value of the output. The state of the circuit is entirely characterized by the set of input and output variables of the gates.

We assume that all circuits are *closed*: Each variable is the input of a gate and the output of a gate. (We shall see that we can ignore self-loops and postulate that a variable is shared by exactly two different gates.) An open circuit is transformed into a closed one by representing the environment of the circuit as gates.

**Definitions and Notations.** *An execution of a simple assignment is called a transition. The result of a transition of type  $x \uparrow$  is the postcondition  $x$ ; the result of a transition of type  $x \downarrow$  is the postcondition  $\neg x$ . The simple assignments  $x := \text{true}$  and  $x := \text{false}$  are denoted by  $x \uparrow$  and  $x \downarrow$ , respectively.*

A gate with output variable  $z$  is defined by the two *production rules* (p.r.'s):

$$\begin{aligned} B_u &\mapsto z \uparrow \\ B_d &\mapsto z \downarrow \end{aligned}$$

where  $B_u$  is the condition on the input variables for a transition  $z \uparrow$  to take place, and  $B_d$  is the condition on the input variables for a transition  $z \downarrow$  to take place— $B_u$  and  $B_d$  are called the *guards* of the p.r.'s. The two production rules of a gate have to fulfil the *non-interference* requirement.

**Non-interference.**  $\neg B_u \vee \neg B_d$  is invariantly true.

The result of a production rule is the result of the transition caused by the execution of a production rule.

All production rules with a true guard are executed concurrently. The execution of a production rule is considered correctly terminated when the result holds. The execution of a p.r. correctly terminates except if the guard is falsified before the result holds. In that case, the net-effect of the execution is undefined. We therefore add a semantic requirement (to be proved invariantly true): *stability* of a guard in a computation.

**Stability.** The guard of a production rule is stable in a computation when it is falsified only in states where the result of the production rule holds.

We exclude *self-invalidating* production rules. A rule with guard  $g$  and result  $r$  is self-invalidating if  $r \Rightarrow \neg g$ , like, for example, the rules  $x \mapsto x \downarrow$  and  $\neg x \mapsto x \uparrow$ .

The execution of a p.r. in a state where the result holds is called *vacuous*, and is called *effective* otherwise. From the definition of the execution of a p.r., the vacuous execution of a p.r. is equivalent to a *skip*. Consequently, it is always possible to modify the guard of a p.r. so that it does not contain the output variable of the gate. (Left as an exercise for the reader.) Hence, we can eliminate self-loops, i.e., variables that are input and output of the same gate. In the sequel, unless specified otherwise, an execution of a p.r. means an effective execution.

## 2. Wires, Forks, and Multiple-Output Gates

A priori, a wire with input  $x$  and output  $y$  is the gate defined by the p.r.'s  $x \mapsto y \uparrow$  and  $\neg x \mapsto y \downarrow$ . But the composition of any gate, including a wire, with a wire is the gate itself with one of its variables renamed. Hence, we can add an arbitrary number of wire gates to a circuit definition without actually changing the circuit. In order to have a unique network of gates for each circuit, we exclude the wire from the repertoire of gates: A wire is just a renaming mechanism for variables.

We also exclude the *fork* from the repertoire of gates. A fork has one input and at least two outputs. The fork  $f$  with input  $x$  and outputs  $y$  and  $z$  is defined by the two p.r.'s  $x \mapsto y \uparrow, z \uparrow$  and  $\neg x \mapsto y \downarrow, z \downarrow$ . The generalization to an arbitrary number of outputs is obvious. The gate

$$\begin{aligned} B_u &\mapsto x \uparrow \\ B_d &\mapsto x \downarrow \end{aligned}$$

composed with fork  $f$  is equivalent to the gate with outputs  $y$  and  $z$

$$\begin{aligned} B_u &\mapsto y \uparrow, z \uparrow \\ B_d &\mapsto y \downarrow, z \downarrow \end{aligned}$$

Hence, the fork is just a mechanism for replicating the outputs of a gate and for defining gates with an arbitrary number of outputs. But gates defined in this way have an important restriction: *The effective execution of a production rule of a gate contains an effective transition on each output of the gate.*

The only restriction on the class of circuits considered that these definitions and conventions introduce is the exclusion of arbitration devices. They do not restrict the delay-insensitivity assumption.

### 3. Partial Order of Transitions

The specification of a circuit defines a partial order of actions taken from a repertoire of commands. In order to assert that a circuit implements a specification, we relate this partial order to some other order relation among transitions of a circuit. And we will say that a circuit implements a specification when the partial order of transitions in each computation of the circuit contains the specification partial order in a way that we will explain later.

Consider an effective execution of a p.r. with term  $C$  of the guard true, and let  $t$  be the transition of this execution. (We assume that the guard is in disjunctive normal form, i.e., it is either a *literal*, or a *term*, or a disjunction of terms. A literal is a variable or its negation, and a term is a conjunction of literals.)

We attach to  $C$  a set  $T$  of transitions in the following way. Each literal of  $C$  uniquely defines a transition: The literal  $x$  is the result of a transition of type  $x \uparrow$ ; the literal  $\neg x$  is the result of a transition of type  $x \downarrow$ . (The initialization of a variable is also considered a transition.) *By definition, we say that transition  $t$  is a successor of each transition of  $T$ .*

From the *successor* relation, we can now construct a relation  $<$  which is a pre-order, i.e., it is transitive and anti-reflexive.

**Transitivity** For any two transitions  $t_1$  and  $t_2$ , we say that  $t_1 < t_2$  when  $t_2$  is a successor of  $t_1$ , or there exists a transition  $t_3$  such that  $t_1 < t_3$  and  $t_3 < t_2$ .

**Anti-reflexivity**  $t < t$  holds for no transition  $t$ .

Anti-reflexivity is satisfied if, for each ring of gates in the circuit, there is always at least one p.r. whose guard is true and whose result is false—the ring “oscillates.” Anti-reflexivity excludes rings of gates that are used to maintain constant values of variables, like in cross-coupled device constructions of storage elements. We therefore assume that the storage elements are parts of “perfect wires,” so to speak, which keep the value of a variable until the next transition on the variable.

Once we have the pre-order relation  $<$ , we construct the partial order  $\leq$  by defining  $t_1 \leq t_2$  to mean  $t_1 < t_2$  or  $t_1 = t_2$ .

**Definition.** A chain from  $a$  to  $b$  is a finite, non-empty set  $\{t_i, 0 \leq i < n\}$  of transitions such that  $t_0 = a$ ,  $t_n = b$ , and for all  $i$ ,  $0 < i < n$ ,  $t_i$  is a successor of  $t_{i-1}$ . By construction,  $a \leq b$  means that there is a chain from  $a$  to  $b$ .

If  $a < b$ , we will sometimes say that  $b$  follows  $a$ .

## 4. Implementation of Stability

Consider again an execution of p.r. with guard  $B$  and transition  $t$ . Either  $B$  is never falsified once it holds, but then  $t$  is the last transition on the variable involved—we say that the transition is *final*. Or  $B$  is falsified after a finite number of transitions following  $t$ . In that case, in order to implement stability, we have to see to it that  $t$  is completed before  $B$  is falsified.

For all transitions  $i$  that falsify  $B$ , we have to guarantee  $t < i$ . Hence, by definition of the order relation, there must be a transition  $s$  such that  $s$  is a successor of  $t$ , and  $s \preceq i$ . We say that  $s$  *acknowledges*  $t$ .

**Acknowledgment Theorem.** *In a d.i. circuit, each non-final transition  $t$  has a successor transition.*

By construction of multiple-output gates, we have the

**Corollary.** *In a d.i. circuit, a non-final transition on an input of a gate has a successor transition on each output of the gate.*

## 5. The Unique-Successor-Set Criterion

Later on, we shall give a simple criterion to decide whether a given circuit—a network of gates—is delay-insensitive. But such a criterion does not tell us whether there exists a d.i. circuit for a given specification. We shall therefore formulate a more general theorem which characterizes the partially ordered sequences of transitions that admit a d.i. implementation. This criterion enables us to decide that a program does not have a d.i. implementation without having to construct a circuit.

A *computation* is a partially ordered sequence of transitions corresponding to a possible execution of a circuit. It is finite if the computation terminates, and infinite otherwise.

**Successor Set.** *In a computation, the successor set of a transition  $t$  is the set of variables  $x$  such that a transition  $tx$  on  $x$  is a successor of  $t$ .*

**Unique-Successor-Set Property.** *A computation has the unique-successor-set (USS) property when all non-final transitions on the same variable have the same successor set. A set of computations has the USS property when all non-final transitions on the same variable have the same successor set in all computations of the set.*

**Unique-Successor-Set Theorem.** *A set of computations of a d.i. circuit has the USS property.*

**Proof.** From the definition of the successor of a transition and the corollary, the successor set of a non-final transition on a variable, say,  $y$ , is the set of output variables of the gate of which  $y$  is an input.



Since this gate is uniquely defined by the circuit topology, the successor set is unique for all transitions on  $y$  in all computations corresponding to an execution of the circuit.  $\square$

Although the Unique Successor Set Theorem is a direct consequence of the Acknowledgment Theorem, its formulation in terms of computations instead of gates makes it possible to lift the result from the implementation level to the specification level. We assume that whatever specification notation is used—programs, traces, regular expressions, temporal logic, etc.—it is possible to derive certain properties of the partial ordering of actions involved from the specification. Hence, in the sequel, a specification means a partially ordered sequence of actions taken from some repertoire of commands.

Since the partially ordered sequence of actions defining the specification is a projection of the sequence of actions implementing it, we shall investigate whether the USS property is maintained by projection.

**Definition.** *Given a computation  $c$  on a set  $V$  of variables, the projection of  $c$  on a subset  $W$  of  $V$  is the computation derived from  $c$  by removing all transitions on variables of  $V \setminus W$  from the chains of  $c$ . The projection of a set of computations is the set obtained by projecting each element of the original set.*

**Projection Theorem.** *If a set of computations has the USS property, then its projection on a subset of variables has the USS property.*

**Proof.** By definition, the projection of a set of computations on  $W$  can be obtained by removing the elements of  $V \setminus W$  one for one from all chains of each computation of the set. We prove the theorem by showing that removing all transitions on one variable, say,  $w$ , maintains the USS property of the set.

Let  $x$  be another variable, and let  $X$  be the USS of (all transitions on)  $x$  in all computations of the set. Either  $w$  does not belong to  $X$  and  $X$  is left unchanged by the transformation. Or  $w$  is removed from  $X$ . But then, for each transition  $tx$  on  $x$ , the successor set of the transition on  $w$  that follows  $tx$  has to be added to the successor set of  $tx$ . Since all transitions on  $w$  have the same successor set in all computations of the set, the new  $X$  is the same for all transitions and all computations of the set.  $\square$

#### EXAMPLE

The cyclic program  $*[X; Y]$  where  $X$  and  $Y$  are communication commands is called a *one-place buffer*. It is a basic building block of asynchronous circuit design. With a four-phase handshaking protocol for implementing the communications, an expansion of the program is in terms of elementary variables is:

$$*[[xi]; xo \uparrow; [\neg xi]; xo \downarrow; yo \uparrow; [yi]; yo \downarrow; [\neg yi]]$$

where  $xi$  and  $yi$  are the input variables, and  $xo$  and  $yo$  are the output variables. The command  $[B]$  is a shorthand notation for  $[B \rightarrow skip]$ , and can be informally described as “wait until  $B$  holds”.

The environment of the circuit can be simply modeled as the two programs:

$$\begin{aligned} &*[xi \uparrow; [xo]; xi \downarrow; [\neg xo]] \\ &*[[yo]; yi \uparrow; [\neg yo]; yi \downarrow]. \end{aligned}$$

The three programs are concurrent. Now observe that the projection of an infinite computation on the input variables of the first program gives the infinite computation described by the program

$$*[xi \uparrow; xi \downarrow; yi \uparrow; yi \downarrow].$$

Obviously, this infinite computation does not have the USS property and, therefore, the closed circuit implementing the three programs is not d.i.. But the two environment programs can be implemented with an inverter and a wire, which are d.i. circuits. Hence, a circuit implementing this version of the one-place buffer is not d.i..  $\square$

## 6. Specifications and the USS Property

The Projection Theorem is very useful because we can also define when a specification has the USS property, so that if a specification does not have the property, we can immediately conclude that there exists no d.i. implementation of the specification. The projection from implementation to specification occurs as follows.

Given is an arbitrary command (or statement)  $S$ ;  $S$  can be of any kind: assignment, communication, procedure call, transition of a finite-state machine, etc. We make the—in theory slightly restrictive—assumption that an elementary variable can be uniquely identified with each command of the repertoire, i.e., the transitions on the variable occur in the executions of the command only, and each execution of the command contains a transition on the variable. (This assumption is needed for the specification theorem only.)

Consider then a specification implying a certain partial order of actions on a given repertoire of commands  $X$ ,  $Y$ ,  $Z$ , etc. This partial order—which we now call the *specification*—implies the same partial order on a set of transitions on the elementary variables  $x$ ,  $y$ ,  $z$ , etc., that can be uniquely attached to the commands.

Hence, the specification defines a projection of the computation on the set of variables  $\{x, y, z, \dots\}$ . According to the Projection Theorem, we can then formulate the following

**Specification Theorem.** *If the specification of a circuit does not have the USS property, the circuit has no d.i. implementation.*

**EXAMPLES** The following examples, which we give without proofs, show how limited is the class of programs that admit a d.i. implementation. (In the examples, all commands are different from the empty command *skip*.) We assume that the

semantics of the program notation is clear enough that we can identify the programs with the partial order of actions they represent.

- Let  $P \equiv *[S_1; S_2; \dots S_n]$ , and assume that there is no equivalent program

$$*[S_1; S_2; \dots S_k]$$

with  $k < n$ . (We say that  $P$  is a minimal representation. For instance,  $*[X; X]$  is not minimal since  $*[X]$  is an equivalent program.)

Then  $P$  has the USS property only if  $S_i \neq S_j$  for  $i \neq j$ . (That the condition is not sufficient is shown by the previous example.) Hence, the “modulo-2 counter”  $*[X; X; Y]$  and all other “modulo- $k$  counters” have no d.i. implementation.

- The program  $*[S_1; [B_1 \rightarrow S_2 \parallel B_2 \rightarrow S_3]; S_4]$ , with  $S_2 \neq S_3$ , does not have the USS property. Hence, there is no d.i. circuit implementing such a selection command.

□

## 7. Gate characterization of d.i. circuits

**Definition.** An  $n$ -input gate in which  $B_u$  is the conjunction of the  $n$  input variables and  $B_d$  is the conjunction of the negations of the  $n$  input variables is called an  $n$ -input C-element. A gate derived from a C-element by negating one or more literals in  $B_u$  or  $B_d$  is also a C-element.

The Muller-C element is a two-input C-element according to our definition. A one-input C-element reduces to either a wire or an inverter.

**C-element Theorem.** If each computation of a d.i. circuit contains at least 3 transitions on each variable, the circuit comprises only C-elements, or gates that can be replaced with C-elements.

**Proof.** Let  $x$  be an arbitrary variable of the circuit;  $x$  is the input of gate  $g$  with output  $z$ . We shall prove that  $g$  can be implemented as a C-element.

We consider an arbitrary computation of the circuit. First, observe that because of the non-interference, all transitions on the same variable are totally ordered. And because all transitions are effective, upgoing and downgoing transitions on the same variable alternate.

Since the circuit contains at least 3 (effective) transitions on each variable, at least one transition of type  $x \uparrow$  is followed by a transition of type  $x \downarrow$ . And at least one transition of type  $x \downarrow$  is followed by a transition of type  $x \uparrow$ .

Let  $t1$  be a transition of type  $x \uparrow$  and  $t2$  the transition of type  $x \downarrow$  following it. For the guard of the p.r. of  $t1$  to be stable, there must be a transition  $tz$  on  $z$  such that  $t1 < tz < t2$ . We also know that  $tz$  is a successor of  $t1$ .

By the USS theorem and the Projection theorem, there is exactly one transition  $tz$  on  $z$  such that  $t1 < tz < t2$ . By the same argument, there is exactly one

transition on  $z$  between a transition of type  $x \downarrow$  and the transition of type  $x \uparrow$  following it.

Without loss of generality, assume that the first transition on  $x$  is of type  $x \uparrow$  and the first transition on  $z$  is of type  $z \uparrow$ . Then, because of the alternation of upgoing and downgoing transitions on each variable, each transition of type  $z \uparrow$  is the successor of a transition of type  $x \uparrow$ . And each transition of type  $z \downarrow$  is the successor of a transition of type  $x \downarrow$ .

By definition of the successor relation, all terms of guard  $B_u$  of  $g$  contain  $x$ . Hence,  $B_u$  is of the form  $x \wedge C_u$ , where  $C_u$  does not contain  $x$ . Symmetrically, guard  $B_d$  of  $g$  is of the form  $\neg x \wedge C_d$ , where  $C_d$  does not contain  $x$ . Since this property of  $B_u$  and  $B_d$  holds for each input of  $g$ ,  $g$  is a C-element or can be replaced with a C-element.  $\square$

## 8. For Whom the Bell Tolls?

Are these results tolling the bell of d.i. design? Actually, not. At worst, they may slightly embarrass those researchers who claim to have a design method for entirely d.i. circuits. At best, they vindicate the compromises to delay-insensitivity adopted by several asynchronous design methods.

The compromise I have introduced is that of *isochronic forks*. In an isochronic fork, the transitions on *all* outputs of the fork are completed when a transition on *one* output has been acknowledged. Hence, some transitions on some outputs of an isochronic fork need not be acknowledged, and thus the Acknowledgment Theorem does not always hold.

The extension of a standard repertoire of d.i. gates with isochronic forks is sufficient to construct any circuit of interest. I believe it is the weakest possible extension in the sense that any other compromise includes isochronic forks.

## 9. Acknowledgments

The formulation of the C-element Theorem in terms of three transitions on each variable is due to a suggestion from Pieter Hazewindus. Acknowledgment is also due to Steve Burns, Peter Hofstee, Marcel van der Goot, Tony Lee, and José Tierno for their comments and criticisms. The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745.



# A Framework for Adaptive Routing in Multicomputer Networks

John Y. Ngai and Charles L. Seitz

Department of Computer Science  
California Institute of Technology\*

## Introduction

Message-passing concurrent computers, also known as *multicomputers*, such as the Caltech Cosmic Cube [1] and its commercial descendents, consist of many computing nodes that interact with each other by sending and receiving messages over communication channels between the nodes [2]. The communication networks of the second-generation machines, such as the Symult Series 2010 and the Intel iPSC2, employ an *oblivious* wormhole routing technique [3,4] that guarantees deadlock freedom. The network performance of this highly evolved oblivious technique has reached a limit of being capable of delivering, under random traffic, a *stable* maximum sustained throughput of  $\approx 45$  to 50% of the limit set by the network bisection bandwidth. Further improvements on these networks will require an *adaptive* utilization of available network bandwidth to diffuse local congestions.

In an adaptive multipath routing scheme, message routes are no longer deterministic, but are continuously perturbed by local message loading. It is expected that such an adaptive control can increase the throughput capability towards the bisection bandwidth limit, while maintaining a reasonable network latency. While the potential gain in throughput is at most only a factor of 2 under random traffic, the adaptive approach offers additional advantages, such as the ability to diffuse local congestions in unbalanced traffic, and the potential to exploit inherent path redundancy in richly connected networks to perform *fault-tolerant* routing. The rest of this paper consists of an examination of the various feasibility issues and results concerning the adaptive ap-

proach studied by the authors. A much more detailed exposition, including results on performance modeling and fault-tolerant routing, can be found in [5].

## The Adaptive Cut-Through Model

It is clear that in order for the adaptive multipath scheme to compete favorably with the existing oblivious wormhole technique, it must employ a switching technique akin to *virtual cut-through* [6]. In cut-through switching and its blocking variant, which is used in oblivious wormhole routing, a packet is forwarded immediately upon receipt of enough header information to make a routing decision. The result is a dramatic reduction in the network latency over the conventional store-and-forward switching technique under light to moderate traffic. We now describe a simple cut-through switching model that provides the context for the discussion of issues involved in performing adaptive routing in multicomputer networks. The following definitions develop the notation that will be used throughout the rest of the paper.

**Definition 1** A *Multicomputer Network*,  $M$ , is a connected undirected graph,  $M = G(N, C)$ . The vertices of the graph,  $N$ , represent the set of computing nodes. The edges of the graph,  $C$ , represent the set of *bidirectional* communication channels.

**Definition 2** Let  $n_i \in N$  be a node of  $M$ . The set,  $C_i \subseteq C$ , is the set of bidirectional channels connecting  $n_i$  to its neighbors in  $M$ .

**Definition 3** The *width*,  $W$ , of a channel is the number of data wires across the channel. A *flit*, or *flow control unit*, is the  $W$  parallel bits of information transferred in a single cycle. The flit is the unit used to measure the length of a packet.

**Definition 4** Given a pair of nodes,  $n_i$  and  $n_j$ , the set,  $Q_{ij}$ , of *routes* joining  $n_i$  to  $n_j$  is the fixed and predetermined set of directed *acyclic* paths from the source node,  $n_i$ , to the destination node,  $n_j$ .

**Definition 5** For each destination node,  $n_j$ , the *profitable* channel set,  $R_{ij} \subseteq C_i$ , is the subset of channels

---

\*The research described in this paper was sponsored in part by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745; and in part by grants from Intel Scientific Computers and Ametek Computer Research Division.

connected to  $n_i$ , where  $c_k \in R_{ij} \Rightarrow c_k \in q_m \in Q_{ij}$ . In other words, forwarding a packet along the routes in  $Q_{ij}$  is equivalent to sending it out through a profitable channel in  $R_{ij}$ .

**Definition 6** For each node,  $n_i \in N$ , the *Routing Relation*  $R_i = \{(n_j, c_k) : n_j \in N - \{n_i\}, c_k \in R_{ij}\}$  defines for each possible destination node,  $n_j \in N$ , its corresponding profitable channel set,  $R_{ij}$ .

**Definition 7** The actual path a packet traverses while in transit in the communication network is referred to as the *trajectory* of the packet. Packet trajectories are identical to the packet routes in oblivious routing schemes but are *non-deterministic* in our adaptive formulation.

We assume the following:

- Long messages are broken into packets that are the logical data entities transferred across the network.
- Packets are of fixed length; i.e., packet length =  $L$ , where  $L$  is a network-wide constant.
- Complete routing information is included in the header flit of each packet.
- Packets are forwarded in virtual cut-through style.
- A message packet arriving at its destination node is consumed. This is commonly known as the *consumption assumption*.
- A node can generate messages destined to any other node in the network.
- Nodes can produce packets at any rate subject to the constraint of available buffer space in the network, and packets are source queued.
- Each node in the network has complete knowledge of its own routing relation.

Figure 1 presents our view of the structure of a node in a multicomputer network. Conceptually, a node can be partitioned into a computation subsystem, a communication subsystem, and a message interface. For our purpose, the computation subsystem serves as the producer and consumer of the messages routed by the communication subsystem of the node. The message interface consists of dedicated hardware that handles the overhead in sending, receiving, and reassembling message packets. Internally, the communication subsystem consists of an adaptive control and a small number of message-packet buffers. Routing decisions are made by the adaptive control, based entirely on locally available information. The bidirectional channel assumption is adopted to allow the network to exploit locality in general message-communication patterns. Furthermore, it

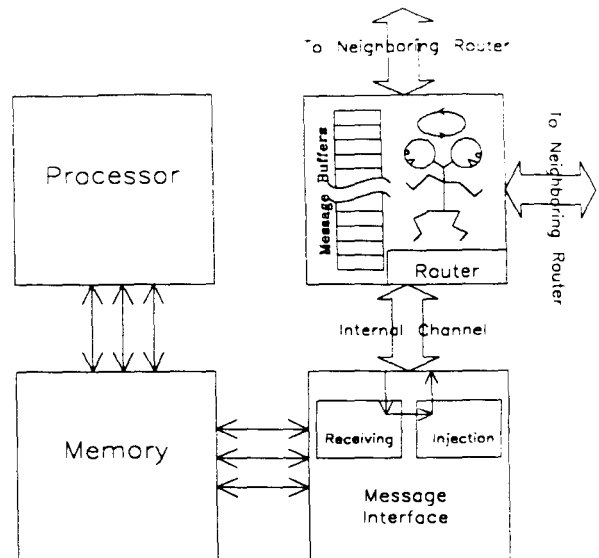


Figure 1: Structure of a node.

assures an identical number of input and output communication channels in each node, irrespective of the underlying network topology. The fixed-packet-length assumption is not essential and can be replaced by a *bounded-packet-length* assumption, i.e., packet length  $\leq L$ , without invalidating any of our major results. It is adopted solely to simplify the exposition.

### Communication Deadlock Freedom

In any adaptive routing scheme that allows arbitrary multipath routing, it is necessary to assure freedom from communication deadlock. Communication deadlock is caused generically by the existence of *cyclic* dependencies among communication resources along the message routes. Methods to prevent communication deadlock have been intensively researched and many schemes exist; of these, the methods of structured buffer pools [7] and virtual channels [8] are representative. In essence, all of these methods approach the problem by re-mapping any dependency that is potentially cyclic into a corresponding *acyclic* dependency structure. These methods employ restructuring techniques that require information of a global, albeit static, character. In contrast, a very simple technique that is independent of network size and topology, using voluntary *misrouting*, was suggested in [9] for networks that employ data exchange operations. Such a preemption technique utilizes only local information, and is dynamic in character. It prevents deadlock by breaking the potentially cyclic communication dependencies into disjoint paths of unit length. Voluntary misrouting can be applied to assure deadlock freedom in cut-through switching networks, provided the input and output data rates across the channels at each node are tightly matched.

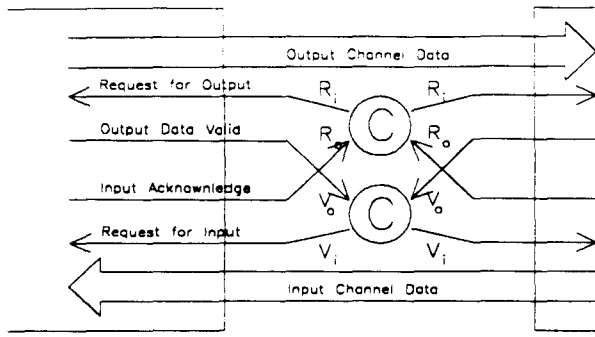


Figure 2: Two-phase protocol signaling.

A simple way is to have all bidirectional channels of the same node operate *coherently* under the protocol described next.

**The Coherent Protocol.** We now describe the channel data-exchange protocol in detail. It is used to match the transfer rates across all channels of the same node. The protocol employs four control signals per channel, two from each of the communicating partners, and is completely symmetric between the partners. The signaling events for a channel,  $c \in C$ , are:

- $R_o$  — output event to the communicating partner indicating that this node is Ready to accept another input flit from its partner. It also serves as an acknowledgment to its partner of the successful completion of the previous transfer cycle.
- $R_i^c$  — input event from the communicating partner indicating that the partner is Ready to accept another output flit from this node. It is also an acknowledgment from the partner of the successful completion of the previous transfer cycle.
- $V_o$  — output event to the communicating partner indicating that the data flit values currently held at the output channel of this node are Valid and its partner should latch in the held values.
- $V_i^c$  — input event from the communicating partner indicating that the data flit values currently asserted at the input channel of this node are Valid and the node should latch in the held values.

We proceed to define our handshaking protocol across channels of a node,  $n_k \in N$ , in a CSP-like notation [10]:

$$*[ \begin{array}{l} R_o, \quad [\forall c \in C_k, R_i^c]; \quad \text{apply out data;} \\ V_o, \quad [\forall c \in C_k, V_i^c]; \quad \text{latch in data;} \end{array} ]$$

Observe that  $R_o$  and  $V_o$  denote, respectively, the *unique* outgoing Ready and data Valid signaling event to all neighbors of  $n_k$ . This enforces the matching of outgoing data rates. On the other hand, the matching of incoming data rates is enforced through the *synchronized*

wait for the  $R_i^c$  and  $V_i^c$  signaling events from all neighbors. The handshaking events,  $R_o$  and  $R_i^c$ , interlock with events  $V_o, V_i^c$  to guarantee the stability and strict alternation of each other. The *initial* state of a channel has both directions of the channel ready to accept a new data flit, and proceeds thereafter in a *demand-driven* fashion. Figure 2 shows a possible conceptual realization of the protocol under the two-phase signaling convention [11] popular for off-chip communication. Since all the handshaking events defined are local between nearest neighbors, a network following the coherent protocol is *arbitrarily extensible*.

Observe that under cut-through switching, a packet can span many different channels. An outgoing channel occupied by a packet may not be able to assert  $V_o$  until after valid data has been asserted by the corresponding incoming channel occupied by the packet; this induces matching of data rates across the two occupied channels. The notion of coherency introduced here is a natural way to accommodate such potential dependencies among the various channels of a node under cut-through switching. Another notion that arises naturally is that of a *null flit*. To effect a transfer of data in one direction of a channel while the opposite direction is idle, the receiving partner is required to transmit a null flit in order to satisfy the convention dictated by the exchange protocol.

**Deadlock Freedom.** We now demonstrate that to assure communication deadlock freedom for networks operating under the coherent protocol, it is sufficient to employ voluntary misrouting to prevent potential buffer overflow. To proceed, observe that routing under the cut-through switching model imposes the following integrity constraints:

1. Packets must always be forwarded to neighbors with their header flits transmitted first. In particular, voluntary misrouting of any internally buffered packet must start from the header flit of the selected packet.
2. Once the flit stream of a packet has been assigned a particular outgoing channel, the assignment must be maintained for the remaining cycles until the entire packet has been transmitted.

These constraints exist because all of the necessary routing information of a packet is encapsulated in the packet header. Interrupting a packet flit stream mid-transfer would render the latter part of the packet undeliverable. To establish deadlock freedom, it is sufficient to show that each node can *independently* complete each transfer cycle and initiate a new one, in a bounded period, without violating the stated constraints. We now

show that as long as we have an equal number of input and output channels per node, a condition satisfied readily by our bidirectional channel assumption, we can always satisfy the stated logical requirements, thereby assuring freedom from communication deadlock.

**Theorem 1** Let  $M$  denote a coherent multicomputer network where each node has an equal number of input and output channels. If  $M$  employs voluntary misrouting to prevent potential buffer overflow, then it is free from deadlock.

**Proof.** We need to show that buffer overflow can always be prevented by misrouting without violating the cut-through switching integrity constraints. We proceed with a counting argument: Let  $d$  denote the number of channels at a node. During a protocol cycle, there may be as many as  $n^* \leq d$  new data flits arriving at the input channels. A fraction of these,  $0 \leq n' \leq n^*$ , are new header flits; the remaining  $n^* - n'$  are non-header flits of arriving packets. Of these non-header flits, a fraction of them,  $0 \leq n'' \leq n^* - n'$ , belong to packets that have already been assigned output channels, and the remaining  $n^* - n' - n''$  flits belong to waiting packets that are buffered inside the node. Therefore, the node has at least a total of  $n' + (n^* - n' - n'')$  header flits that are eligible for immediate routing. Hence, in the following cycle, a node can find at least  $n' + (n^* - n' - n'') + n'' = n^*$  flits that can be transmitted or misrouted without violating the cut-through switching-integrity constraints. This assures that no buffer overflow will occur. Since the node can always complete its protocol cycles in bounded time, the network is free from deadlock. ■

Since the validity of the above proof does not depend on a node's storage capacity, deadlock freedom is established independent of the amount of available buffer space. The simple criterion of having an equal number of input and output channels is sufficient to assure deadlock freedom for a coherent network. In practice, additional buffers are needed in order to inject packets into the network, and to improve the network performance.

#### Network Progress Assurance

The adoption of voluntary misrouting renders communication deadlock a non-issue. However, misrouting also creates the burden of having to demonstrate progress in the form of message-delivery assurance. In particular, a network can run into a *livelock*. Consider the sequence of routing scenarios depicted in figure 3 for a bidirectional ring consisting of eight nodes and eight packets. Each of the packets consists of four data flits that span multiple channels and internal buffers. Suppose the nodes employ the following simple, deterministic, packet-to-channel assignment rule: Whenever two

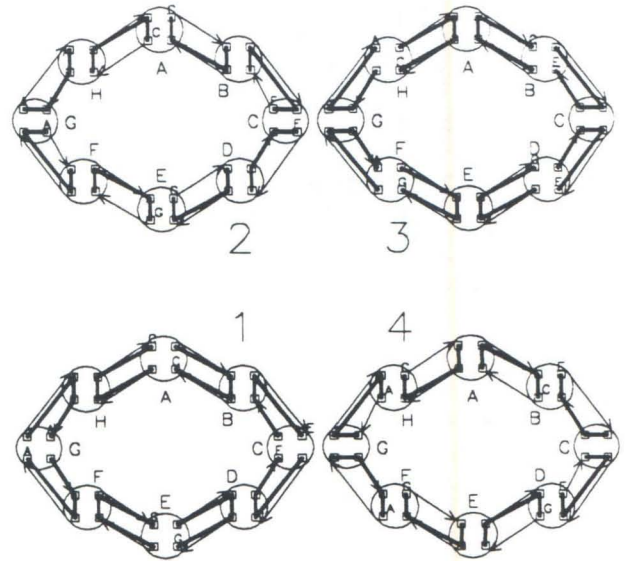


Figure 3: Livelock due to bad assignments.

incoming packets both request the same outgoing channel, the packet from the clockwise neighbor always wins. Given that, initially, nodes A, C, E, and G each receive two packets destined to nodes that are, respectively, distance two from them in the clockwise direction, after four routing cycles, the packets are all back to where they started! This example illustrates that packets can be forever denied delivery to their destinations even in the absence of communication deadlock.

Channel-access competitions are, however, not the only type of conflict that can lead to livelock. Consider the situations depicted in figure 4 for the same bidirectional ring network. The traffic patterns are coincidental in such a way that none of the packets will ever have a chance to select its own output channel; rather, at every node, each packet must be forwarded along the only remaining channel, in compliance with the voluntary misrouting discipline, in order to avoid deadlock. It is clear that no matter what assignment strategy one chooses, it is impossible to break this kind of livelock without adding extra buffers per node. In other words, additional measures and resources have to be introduced in order to assure progress, i.e., delivery of packets, in the network.

**Buffering Discipline and Requirement.** In order to assure packet delivery in spite of voluntary misrouting, extra buffers are required to store packets temporarily. In particular, sufficient buffers must be provided to allow the adaptive control to give *any* newly arriving packet a chance to escape preemption if so determined by the assignment algorithm. We now demonstrate the existence of such a solution using a bounded number of



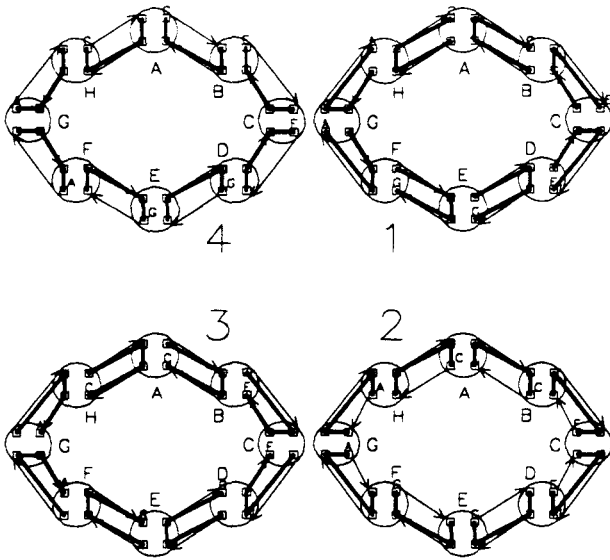


Figure 4: Livelock due to lack of assignments.

buffers. We assume the following buffering discipline:

1. Storage is divided into buffers of equal size; each is capable of holding an entire message packet.
2. Each buffer has exactly one input and one output port; this permits simultaneous reading and writing. A good example is a *FIFO* queue of length  $L$ .
3. Except as stated below, a buffer can be occupied by only one packet at a time. Oftentimes a packet may not fill its entire buffer, as in case of a partial cut-through. Such a packet occupies both the input and output ports to the buffer.
4. A buffer can be used temporarily to store two packets at a time, if and only if, one of them is leaving through the output port connected to an output channel, and the other is entering through the input port connected to an input channel.

Let  $b$  and  $d$  denote, respectively, the number of buffers and channels, i.e., the *degree*, at each node. First, we observe that given the above buffering discipline, we must have  $b \geq d$ . To see this, assume that  $L \gg d$ , and consider the following sequence of events at a node with all buffers initially empty: At cycle  $t = 0$ , a packet  $P_0$  arrives and is forwarded to its requested output channel  $c^*$  at cycle  $t = 1$ . Then, at cycles  $t = L - d$  up to  $t = L - 2$ , a total of  $d - 1$  packets,  $P_i$ ,  $i = 1, \dots, d - 1$ , arrive one after another in  $d - 1$  consecutive cycles, all requesting the same output channel  $c^*$ . Finally, at cycle  $t = L + 2$ , another packet  $P_d$  arrives, requesting the same channel  $c^*$ . The worst case happens when the

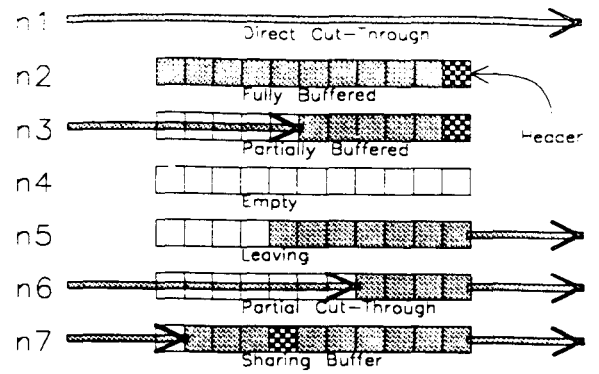


Figure 5: Accounting of buffer allocations.

assignment algorithm always favors the latest-arriving packet by requiring it to stay and avoid preemption, and has each occupy a distinct buffer. Given the above arrival sequence, at cycle  $t = L + 1$ , packet  $P_{d-1}$  will be forwarded through  $c^*$ , which now becomes idle. As a result, each packet from  $P_1$  up to  $P_d$  will have to be temporarily stored as it arrives. Since each packet must be allocated to a distinct buffer, we must have  $b \geq d$ . We now show that having  $b = d$  buffers is also sufficient.

**Theorem 2** Let  $M$  be a coherent network where each node has  $b$  packet buffers inside the router operating under the stated assumptions. Then  $b = d$  buffers per router is necessary and sufficient to always allow at least one packet, chosen arbitrarily by the assignment algorithm at each node, to escape preemption.

**Proof.** Necessity follows immediately from the preceding discussion. We proceed to establish sufficiency through a counting argument. Observe that a node is required to consider misrouting of packets in the next cycle only when there are new packets arriving at the current cycle. Figure 5 depicts an accounting of all possible cases of buffer allocation at the end of any such routing cycle. Let  $n_1$  up to  $n_7$  denote, respectively, the number of packets or buffers in each case; and  $n_0$  denote the number of newly arrived packets. Then, for inputs, we have  $n_0 + n_1 + n_3 + n_6 + n_7 \leq d$ ; for outputs, we have  $n_1 + n_5 + n_6 + n_7 \leq d$ . To simplify the counting argument, let us assume for the moment that  $n_0 = 1$ . Let  $P^*$  denote the privileged packet chosen by the assignment algorithm to stay behind and avoid misrouting in the following cycle.  $P^*$  must be either a newly arrived packet or an already buffered packet. If  $P^*$  is a buffered packet, then either the newly arriving packet finds an idle output channel to directly cut through the node, or else we must have  $n_1 + n_5 + n_6 + n_7 = d \Rightarrow n_5 \geq n_0 + n_3$ , which, in turn, implies that there will always be an available buffer ready to accept it. On the other hand, if  $P^*$  is a newly arriv-



ing packet, then either  $n_4 + n_5 > 0$ , and, hence, there is a buffer ready to accept it, or else we must have  $n_2 + n_3 + n_6 + n_7 = b = d$ . This, together with the above inequality on inputs,  $\Rightarrow n_2 \geq n_0 + n_1 \Rightarrow n_2 > 0$ . Furthermore,  $n_0 > 0 \Rightarrow n_1 + n_6 + n_7 < d$ . In other words, the packet will be able to find at least one buffer with a full idle packet as well as an idle output channel to preempt this idle packet and thus make room for itself. This establishes the validity for single-packet arrivals. Finally, repeated applications of the above argument then establish the validity for multiple packet arrivals, and, therefore, the sufficiency condition. ■

The trick in allowing the escape from misrouting for any arbitrarily chosen packet is to provide at least a critical, minimum number of buffers that is sufficient to assure either that empty buffers still exist, or that all buffers have been occupied, and, hence, that there is some other packet that can be misrouted instead. The particular number required depends on the adopted buffering structure and discipline; adding more buffers per node will allow the assignment algorithm to operate with more flexibility and perform better. In any case, by having a sufficient number of buffers, competition of profitable channel access is transformed into a competition for the right to stay behind and wait until the winner's profitable channel becomes available, at which time it will be forwarded. Hence, winners chosen by the assignment algorithm will have the chance to follow the actual paths determined by the routing relations. In a sense, assurance of packet delivery has now been reduced to that of picking *consistent* winners across the network.

**Packet-Priority Assignments.** An effective scheme for picking consistent winners that is independent of any particular network topology is to resolve the channel-access conflicts according to a *priority* assignment. In particular, the process of forwarding a packet towards its destination can be viewed as a sequence of actions performed to reduce the packet's distance from destination, provided that the set  $\mathcal{R} = \{R_i\}$  of routing relations is defined in terms of an underlying metric of the network. In this case, as the result of a channel-access conflict, the winner will be routed along a profitable channel, thus decreasing its distance from the destination. The losers, depending on whether or not they are misrouted along the remaining unprofitable channels, may or may not increase their distance from destination. Ideally, one would prefer a strict monotonic decrease of distance to destination for each packet routed in the network. As this is impossible under our adaptive model, the alternative is to ensure monotonic decrease over a sequence of exchanges involving multiple packets. This can be achieved by giving higher priority to

packets with shorter distances from destination than to those with longer distances, as follows:

$$P_1 > P_2 \iff D_1 < D_2,$$

where  $P$  is a packet's priority and  $D$  its distance from destination. We now show that this is sufficient to guarantee livelock freedom.

**Theorem 3** A packet-to-channel assignment strategy that observes the defined distance priority, together with the set  $\mathcal{R}$  of metric-based routing relations, guarantees livelock freedom in a network.

**Proof.** At the beginning of a routing cycle, let  $D > 0$  be the minimum packet distance from destination. During this cycle, a packet with distance  $D$  competes with other packets for channels leading to its destination. If it wins the competition, it will be forwarded along a profitable channel within  $L$  cycles. If it loses, it must be to another packet also distance  $D$  away from its destination, according to the defined priority. In both cases, the minimum distance is reduced to  $< D$  within  $L$  cycles. Therefore,  $D$  will eventually be reduced to zero, in which case a successful packet delivery occurs and the above argument can be applied again to assure repeated deliveries. This establishes livelock freedom. ■

Observe that although the distance priority alone suffices to guarantee global progress in a message network, no corresponding statement can be made concerning each individual packet. This is because it is possible for packets that are far away from their destinations to be repeatedly defeated by newly injected packets that are closer to their respective destinations. A more complex priority scheme that assures delivery of *every* packet can be obtained by augmenting the above simple scheme with *age* information, with higher priorities assigned to older packets:

$$(A_1, D_1) > (A_2, D_2) \iff (A_1 > A_2) \vee ((A_1 = A_2) \wedge (D_1 < D_2)),$$

where  $A$  is a packet's *age*, that is, the number of routing cycles elapsed since the injection of the packet. Empirical simulation results indicate that the simple distance-assignment scheme is sufficient for almost all situations, except under an extremely heavy applied load.

#### Network-Access Assurance

A different kind of progress assurance that requires demonstration under our adaptive formulation is the ability of a node to inject packets eventually. Because of the requirement to maintain strict balance of input and output data rates, a node located in the center of heavy traffic might be denied access to the network indefinitely. Figure 6 depicts a possible conceptual realization of a message interface. Its operation is similar to the register insertion ring interface described in

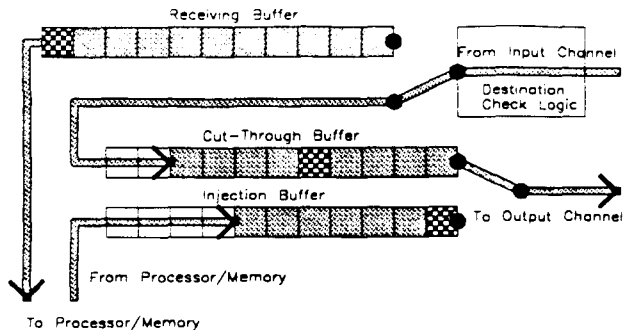


Figure 6: Inside the message interface.

[12]. It uses two FIFO buffers that can be connected to the output channel towards the network via a switch. Whenever the node has a packet to transmit, it loads the packet into the injection buffer as soon as the buffer becomes empty. When message traffic arrives from the network input channel, it passes through the destination check logic, which redirects any traffic destined to this node to the node memory. Any remaining passing traffic is loaded into the cut-through buffer, which is normally connected to the output channel. Whenever the cut-through buffer becomes empty, the control logic checks to see if there is an output packet waiting for injection. In such case, the switch is toggled so that the output channel is connected to the injection buffer, and the injection proceeds. As the output packet is being forwarded, any passing traffic is loaded into the cut-through buffer. The switch connection is flipped back to the cut-through buffer after injection has been finished, and the process repeats. The main interesting property of the message interface for our current discussion is that it provides the mechanism to capture and accumulate interpacket gaps, which need not be contiguous, as empty spaces inside the cut-through buffers. When enough space has been collected, *ie*, the entire packet length, hence, an entire empty buffer, another new packet can be injected into the network. With such a mechanism, the question of assuring eventual packet injection is translated into that of assuring arrival of enough interpacket gaps whenever a node has a packet injection outstanding.

**Round-Trip Packets.** One simple way to assure network access is to have each packet delivered by the network be returned to its original sender upon arrival at its destination. Since each message interface starts with an empty injection buffer, consumption of its own round-trip packets will always restore its ability to inject the next source-queued packet. More sophisticated versions of such a scheme will use several cut-through buffers, and will demand that packets be returned only if the stock of empty cut-through buffers has been depleted

below a predetermined threshold. In this way, the number of round-trip packets can be dramatically reduced when traffic is relatively moderate. Unfortunately, as traffic density increases, the population of round-trip packets also increases, thus further decreasing useful network bandwidth.

**Packet-Injection Control.** A different scheme that does not incur this overhead is to have the nodes maintain a bounded synchrony with neighbors on the total number of injections. Nodes that fall behind will, in effect, prohibit others from injecting until they catch up. We shall adopt the convention that a node having no packet to inject has a *null* packet queued up; *ie*, during each routing cycle, every node either has a null or real packet ready to inject or else is in the process of injecting a real packet. The null-packet convention is required to prevent quiescent nodes that do not have any packet to inject from blocking injections in the active nodes. Our scheme is to introduce *local synchronization* among neighboring nodes such that the total number of packets injected by a node after each routing cycle will not differ by more than  $K$ , a positive constant, from those of its neighbors. We assume that each node explicitly maintains records of the total number of packet injections made by each of its neighbors, measured *relative to that of its own*, and that the information required to update these records in each node is exchanged on separate direct links between the message interfaces among neighbors. A node is allowed to inject its queued packet only if its own number of total injections is fewer than  $K$  packet injections ahead of its minimum neighbor. Nodes that are allowed to inject will examine their queued packets. Null packets are always injected by convention, whereas real packets are injected only if the injection mechanism described previously finds at least one empty buffer available to absorb the injection transient. We now show that, with eventual delivery of the packets already injected, this injection synchronization protocol establishes cooperation among the nodes to assure the eventual occurrence of empty cut-through buffers in the message interface for nodes that have real packets waiting for injection as permitted by the protocol.

**Lemma 4** A node that has a packet waiting for injection that is permissible under the above injection protocol will eventually inject.

**Proof.** Observe that, by convention, if the pending packet is null, the node is able to inject immediately, so that the lemma is true vacuously. We now proceed to establish its validity for real packets. Suppose, to the contrary, that a particular node,  $n \in N$ , is blocked from injection indefinitely because the injection mechanism cannot accumulate sufficient empty buffer space

to absorb the injection transient. Our injection protocol then dictates that its neighbors also will be blocked indefinitely from injecting. These, in turn, indefinitely block their neighbors, and so on. Given a finite network, all nodes are eventually blocked from any further injection, and eventually *no* new packet can enter the network. Given the eventual delivery guarantee for packets already injected, ultimately the network will be void of packets; at that point, the input channel to the interface of  $n$  will become idle, thus enabling it to resume the accumulation of empty spaces inside the cut-through buffer. Eventually, it will have collected enough spaces to enable the injection of its queued packet into the network. This contradicts the original indefinite blocking assumption of  $n$ , and thereby establishes the validity of the lemma. ■

We are now ready to show that by following the above injection protocol every individual node will eventually be permitted to inject, and, hence, according to the above lemma, all will eventually inject. Specifically, let  $M$  be a network, and let  $T_i$  denote the total number of packet injections from node  $n_i \in N$  since initialization. We now prove that  $T_i$  is strictly increasing over time.

**Theorem 5** Given the injection protocol and a finite network that is livelock free, the total number of packet injections for each node strictly increases over time.

**Proof.** During a routing cycle, let  $t = \min_{n_i \in N} T_i$  denote the minimum among numbers of packet injections since initialization, taken over all the nodes of the network, and let  $S = \{n_i \in N | T_i = t\}$  denote the set of nodes that have recorded the minimum number of packet injections since initialization. Since  $K > 0$ , according to our protocol, every node  $n \in S$  is permitted to inject. Lemma 4 then guarantees eventual injections from all of the nodes in  $S$ ; hence,  $t$ , the minimum number of packet injections per node, is guaranteed to eventually increase over time. This, in turn, guarantees that  $T_i$  strictly increases over time,  $\forall n_i \in N$ . ■

Hence, we are assured of eventual packet injection for each individual node of the network. In other words, the above theorem establishes *fairness* in network access among all the nodes.

### Performance Comparisons

An extensive set of simulations was conducted to obtain information concerning the potential gain in performance by switching from the oblivious wormhole to the adaptive cut-through technique. We now summarize very briefly the typical kind of behaviors observed in these simulations. A much more detailed discussion can be found in [5]. Among the various statistics collected, the two most important performance metrics in communication networks are *network throughput* and *message*

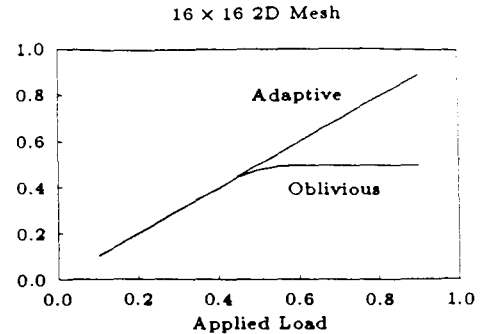


Figure 7: Throughput versus applied load.

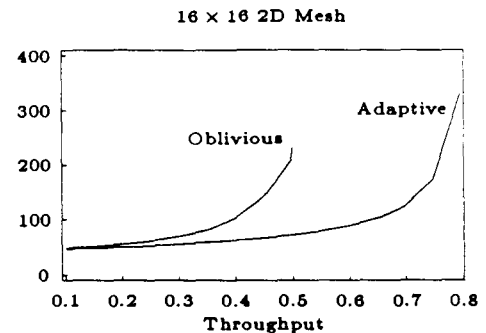


Figure 8: Message latency versus throughput.

*latency*. Figure 7 plots the sustained normalized network throughput versus the normalized applied load of the oblivious and adaptive schemes for a  $16 \times 16$  2D-mesh network under random traffic. The normalization is performed with respect to the network bisection bandwidth limit. Starting at a very low applied load, the throughput curves of both schemes rise along a unit slope line. The oblivious wormhole curve levels off at  $\approx 45 - 50\%$  of normalized throughput but remains *stable* even under an increasingly heavy applied load. In contrast, the adaptive cut-through curve keeps rising along the unit slope until it is out of the range of collected data. It should be pointed out, however, that the increase in throughput obtained is also partly due to the extra silicon area invested in buffer storage, which makes adaptive choices available.

Figure 8 plots the message latency versus normalized throughput for the same 2D-mesh network for a typical message length of 32 flits. The curves shown are typical of latency curves obtained in virtual cut-through switching [6]. Both curves start with latency values close to the ideal at very low throughput, and remain relatively flat until they hit their respective transition points, after which both rise rapidly. The transition points are  $\approx 40\%$  and  $70\%$ , respectively, for the obli-

ous and adaptive schemes. In essence, adaptive routing control increases the quantity of routing service, *ie*, network throughput, without sacrificing the quality of the provided service, *ie*, message latency, at the expense of requiring more silicon area.

### Summary

Several issues related to adaptive cut-through routing have been addressed in the course of this research, and we did not encounter any insurmountable problem. Rather, the simplicity of these resolution mechanisms gives us hope that the adaptive scheme can be made to improve on the already highly evolved oblivious routing scheme. The discussion in this paper has focused on issues concerning the *feasibility* of the proposed adaptive routing framework. Within this framework, we also have studied and found promising approaches to fault-tolerant routing. Clearly, more work remains to be done. Perhaps the most challenging of all is to realize on *silicon* the set of ideas outlined in this study.

### References

- [1] Charles L. Seitz, "The Cosmic Cube," *CACM*, 28(1): 22-33, January 1985.
- [2] William C. Athas and Charles L. Seitz., "Multi-computers: Message-Passing Concurrent Computers," *IEEE Computer*: 9-24, August 1988.
- [3] William J. Dally and Charles L. Seitz, "The Torus Routing Chip," *Distributed Computing* (1): 187-196, 1986.
- [4] Charles M. Flaig, *VLSI Mesh Routing Systems*. Caltech Computer Science Department Technical Report, 5241:TR:87.
- [5] John Y. Ngai, *Adaptive Routing in Multicomputer Networks*. Ph.D. Thesis, Computer Science Department, Caltech. To be published.
- [6] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks* 3(4): 267-286, Sept. 1979.
- [7] P. Merlin, and P. Schweitzer, "Deadlock Avoidance in Store-and-Forward Networks — I: Store-and Forward Deadlock," *IEEE Transactions on Communications*, Vol. COM-28(3): 345-354, March 1980.
- [8] William J. Dally and Charles L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, Vol. C-36(5): 547-553, May 1987.
- [9] A. Borodin, and J. Hopcroft, "Routing, Merging, and Sorting on Parallel Models of Computation," *Journal of Computer and System Sciences* 30: pp. 130-145, 1985.
- [10] Alain J. Martin, "A Synthesis Method for Self-timed VLSI Circuits," *Proc. 1987 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, IEEE Comp. Soc. Press: 224-229, 1987.
- [11] Charles L. Seitz, "System Timing," *Introduction to VLSI Systems*, Chapter 7, C. Mead & L. Conway, Addison-Wesley, 1980.
- [12] M. T. Liu, "Distributed Loop Computer Networks," *Advances in Computers*, M. Yovits, Academic Press: 163-221, 1978.

California Institute of Technology  
Computer Science Department, 256-80  
Pasadena CA 91125

Technical Reports  
18 October 1989

*Prices include postage and help to defray our printing and mailing costs.*

**Publication Order Form**

To order reports fill out the last page of this publication form. *Prepayment* is required for all materials. Purchase orders will not be accepted. All foreign orders must be paid by international money order or by check for a minimum of \$50.00 drawn on a U.S. bank in U.S. currency, payable to CALTECH.

- 
- \_\_\_CS-TR-89-11 \$9.00 *Reactive-Process Programming and Distributed Discrete-Event Simulation*, PhD Thesis  
Su, Wen-King
- \_\_\_CS-TR-89-10 \$7.00 *Silicon Models of Early Audition*, PhD Thesis  
Lazarro, John
- \_\_\_CS-TR-89-09 \$15.00 *A Framework for Adaptive Routing in Multicomputer Networks*, PhD Thesis  
Ngai, John
- \_\_\_CS-TR-89-07 \$6.00 *Constraint Methods for Neural Networks and Computer Graphics*, PhD Thesis  
Platt, John
- \_\_\_CS-TR-89-06 \$1.00 *The First Asynchronous Microprocessor: The Test Results*  
Martin, Alain J, Steven M Burns, T K Lee, Drazen Borkovic, and Pieter J Hazewindus
- \_\_\_CS-TR-89-05 \$2.00 *The Essence of Distributed Snapshots*  
Chandy, K. Mani
- \_\_\_CS-TR-89-04 \$5.00 *Submicron Systems Architecture Project*  
ARPA Semiannual Technical Report
- \_\_\_CS-TR-89-03 \$3.00 *Feature-oriented Image Enhancement with Shock Filters, I*  
Rudin, Leonid I with Stanley Osher
- \_\_\_CS-TR-89-02 \$3.00 *Design of an Asynchronous Microprocessor*  
Martin, Alain J
- \_\_\_CS-TR-89-01 \$4.00 *Programming in VLSI From Communicating Processes to Delay-insensitive Circuits*,  
Martin, Alain J
- \_\_\_CS-TR-88-22 \$2.00 *Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm*,  
Su, Wen-King and Charles L Seitz
- \_\_\_CS-TR-88-21 \$3.00 *Winner-Take-All Networks of  $O(N)$  Complexity*,  
Lazzaro, John, with S Ryckebusch, M A Mahowald and C A Mead
- \_\_\_CS-TR-88-20 \$7.00 *Neural Network Design and the Complexity of Learning*,  
Judd, J Stephen
- \_\_\_CS-TR-88-19 \$5.00 *Controlling Rigid Bodies with Dynamic Constraints*,  
Barzel, Ronen
- \_\_\_CS-TR-88-18 \$3.00 *Submicron Systems Architecture Project*,  
ARPA Semiannual Technical Report
- \_\_\_CS-TR-88-17 \$3.00 *Constrained Differential Optimization for Neural Networks*,  
Platt, John C and Alan H Barr
- \_\_\_CS-TR-88-16 \$3.00 *Programming Parallel Computers*,  
Chandy, K Mani
- \_\_\_CS-TR-88-15 \$13.00 *Applications of Surface Networks to Sampling Problems in Computer Graphics*, PhD Thesis  
Von Herzen, Brian
- \_\_\_CS-TR-88-14 \$2.00 *Syntax-directed Translation of Concurrent Programs into Self-timed Circuits*  
Burns, Steven M and Alain J Martin
- \_\_\_CS-TR-88-13 \$2.00 *A Message-Passing Model for Highly Concurrent Computation*,  
Martin, Alain J



Caltech Computer Science Technical Reports

___CS-TR-88-12	\$4.00	<i>A Comparison of Strict and Non-strict Semantics for Lists</i> , MS Thesis Burch, Jerry R
___CS-TR-88-11	\$5.00	<i>A Study of Fine-Grain Programming Using Cantor</i> , MS Thesis Boden, Nanette J
___CS-TR-88-10	\$3.00	<i>The Reactive Kernel</i> , MS Thesis Seizovic, Jacov
___CS-TR-88-07	\$3.00	<i>The Hexagonal Resistive Network and the Circular Approximation</i> , Feinstein, David I
___CS-TR-88-06	\$3.00	<i>Theorems on Computations of Distributed Systems</i> , Chandy, K Mani
___CS-TR-88-05	\$3.00	<i>Submicron Systems Architecture</i> ARPA Semiannual Technical Report
___CS-TR-88-04	\$3.00	<i>Cochlear Hydrodynamics Demystified</i> Lyon, Richard F and Carver A Mead
___CS-TR-88-03	\$4.00	<i>PS: Polygon Streams: A Distributed Architecture for Incremental Computation Applied to Graphics</i> , MS Thesis Gupta, Rajiv
___CS-TR-88-02	\$4.00	<i>Automated Compilation of Concurrent Programs into Self-timed Circuits</i> , MS Thesis Burns, Stephen M
___CS-TR-88-01	\$3.00	<i>C Programmer's Abbreviated Guide to Multicomputer Programming</i> , Seitz, Charles, Jakov Seizovic and Wen-King Su
___5258:TR:88	\$3.00	<i>Submicron Systems Architecture</i> ARPA Semiannual Technical Report
___5256:TR:87	\$2.00	<i>Synthesis Method for Self-timed VLSI Circuits</i> , Martin, Alain current supply only: see <i>Proc. ICCD'87: 1987 IEEE Int'l. Conf. on Computer Design</i> , 224-229, Oct'87
___5253:TR:88	\$2.00	<i>Synthesis of Self-Timed Circuits by Program Transformation</i> , Burns, Steven M and Alain J Martin
___5251:TR:87	\$2.00	<i>Conditional Knowledge as a Basis for Distributed Simulation</i> , Chandy, K. Mani and Jay Misra
___5250:TR:87	\$10.00	<i>Images, Numerical Analysis of Singularities and Shock Filters</i> , PhD Thesis Rudin, Leonid Iakov
___5249:TR:87	\$6.00	<i>Logic from Programming Language Semantics</i> , PhD Thesis Choo, Young-il
___5247:TR:87	\$6.00	<i>VLSI Concurrent Computation for Music Synthesis</i> , PhD Thesis Wawrzynek, John
___5246:TR:87	\$3.00	<i>Framework for Adaptive Routing</i> Ngai, John Y and Charles L. Seitz
___5244:TR:87	\$3.00	<i>Multicomputers</i> Athas, William C and Charles L Seitz
___5243:TR:87	\$5.00	<i>Resource-Bounded Category and Measure in Exponential Complexity Classes</i> , PhD Thesis Lutz, Jack H
___5242:TR:87	\$8.00	<i>Fine Grain Concurrent Computations</i> , PhD Thesis Athas, William C.
___5241:TR:87	\$3.00	<i>VLSI Mesh Routing Systems</i> , MS Thesis Flaig, Charles M
___5240:TR:87	\$2.00	<i>Submicron Systems Architecture</i> ARPA Semiannual Technical Report
___5239:TR:87	\$3.00	<i>Trace Theory and Systolic Computations</i> Rem, Martin
___5238:TR:87	\$7.00	<i>Incorporating Time in the New World of Computing System</i> , MS Thesis Poh, Hean Lee

Caltech Computer Science Technical Reports

___5236:TR:86	\$4.00	<i>Approach to Concurrent Semantics Using Complete Traces</i> , MS Thesis Van Horn, Kevin S.
___5235:TR:86	\$4.00	<i>Submicron Systems Architecture</i> ARPA Semiannual Technical Report
___5234:TR:86	\$3.00	<i>High Performance Implementation of Prolog</i> Newton, Michael O
___5233:TR:86	\$3.00	<i>Some Results on Kolmogorov-Chaitin Complexity</i> , MS Thesis Schweizer, David Lawrence
___5232:TR:86	\$4.00	<i>Cantor User Report</i> Athas, W.C. and C. L. Seitz
___5230:TR:86	\$24.00	<i>Monte Carlo Methods for 2-D Compaction</i> , PhD Thesis Mosteller, R.C.
___5229:TR:86	\$4.00	<i>anaLOG - A Functional Simulator for VLSI Neural Systems</i> , MS Thesis Lazzaro, John
___5228:TR:86	\$3.00	<i>On Performance of k-ary n-cube Interconnection Networks</i> , Dally, Wm. J
___5227:TR:86	\$18.00	<i>Parallel Execution Model for Logic Programming</i> , PhD Thesis Li, Pey-yun Peggy
___5223:TR:86	\$15.00	<i>Integrated Optical Motion Detection</i> , PhD Thesis Tanner, John E.
___5221:TR:86	\$3.00	<i>Sync Model: A Parallel Execution Method for Logic Programming</i> Li, Pey-yun Peggy and Alain J. Martin current supply only: see <i>Proc SLP'86 3rd IEEE Symp on Logic Programming Sept '86</i>
___5220:TR:86	\$4.00	<i>Submicron Systems Architecture</i> ARPA Semiannual Technical Report
___5215:TR:86	\$2.00	<i>How to Get a Large Natural Language System into a Personal Computer</i> , Thompson, Bozena H. and Frederick B. Thompson
___5214:TR:86	\$2.00	<i>ASK is Transportable in Half a Dozen Ways</i> , Thompson, Bozena H. and Frederick B. Thompson
___5212:TR:86	\$2.00	<i>On Seitz' Arbiter</i> , Martin, Alain J
___5210:TR:86	\$2.00	<i>Compiling Communicating Processes into Delay-Insensitive VLSI Circuits</i> , Martin, Alain current supply only: see <i>Distributed Computing</i> v 1 no 4 (1986)
___5207:TR:86	\$2.00	<i>Complete and Infinite Traces: A Descriptive Model of Computing Agents</i> , van Horn, Kevin
___5205:TR:85	\$2.00	<i>Two Theorems on Time Bounded Kolmogorov-Chaitin Complexity</i> , Schweizer, David and Yaser Abu-Mostafa
___5204:TR:85	\$3.00	<i>An Inverse Limit Construction of a Domain of Infinite Lists</i> , Choo, Young-Il
___5202:TR:85	\$15.00	<i>Submicron Systems Architecture</i> , ARPA Semiannual Technical Report
___5200:TR:85	\$18.00	<i>ANIMAC: A Multiprocessor Architecture for Real-Time Computer Animation</i> , PhD thesis Whelan, Dan
___5198:TR:85	\$8.00	<i>Neural Networks, Pattern Recognition and Fingerprint Hallucination</i> , PhD thesis Mjolsness, Eric
___5197:TR:85	\$7.00	<i>Sequential Threshold Circuits</i> , MS thesis Platt, John
___5195:TR:85	\$3.00	<i>New Generalization of Dekker's Algorithm for Mutual Exclusion</i> , Martin, Alain J current supply only: see <i>Information Processing Letters</i> , <b>23</b> , 295-297 (1986)
___5194:TR:85	\$5.00	<i>Sneptree - A Versatile Interconnection Network</i> , Li, Pey-yun Peggy and Alain J Martin

# Caltech Computer Science Technical Reports

___5193:TR:85	\$2.00	<i>Delay-insensitive Fair Arbiter</i> Martin, Alain J
___5190:TR:85	\$3.00	<i>Concurrency Algebra and Petri Nets</i> , Choo, Young-il
___5189:TR:85	\$10.00	<i>Hierarchical Composition of VLSI Circuits</i> , PhD Thesis Whitney, Telle
___5185:TR:85	\$11.00	<i>Combining Computation with Geometry</i> , PhD Thesis Lien, Sheue-Ling
___5184:TR:85	\$7.00	<i>Placement of Communicating Processes on Multiprocessor Networks</i> , MS Thesis Steele, Craig
___5179:TR:85	\$3.00	<i>Sampling Deformed, Intersecting Surfaces with Quadtrees</i> , MS Thesis, Von Herzen, Brian P.
___5178:TR:85	\$9.00	<i>Submicron Systems Architecture</i> , ARPA Semiannual Technical Report
___5174:TR:85	\$7.00	<i>Balanced Cube: A Concurrent Data Structure</i> , Dally, William J and Charles L Seitz
___5172:TR:85	\$6.00	<i>Combined Logical and Functional Programming Language</i> , Newton, Michael
___5168:TR:84	\$3.00	<i>Object Oriented Architecture</i> , Dally, Bill and Jim Kajiya
___5165:TR:84	\$4.00	<i>Customizing One's Own Interface Using English as Primary Language</i> , Thompson, B H and Frederick B Thompson
___5164:TR:84	\$13.00	<i>ASK French - A French Natural Language Syntax</i> , MS Thesis Sanouillet, Remy
___5160:TR:84	\$7.00	<i>Submicron Systems Architecture</i> , ARPA Semiannual Technical Report
___5158:TR:84	\$6.00	<i>VLSI Architecture for Sound Synthesis</i> , Wawrzynek, John and Carver Mead
___5157:TR:84	\$15.00	<i>Bit-Serial Reed-Solomon Decoders in VLSI</i> , PhD Thesis Whiting, Douglas
___5147:TR:84	\$4.00	<i>Networks of Machines for Distributed Recursive Computations</i> , Martin, Alain and Jan van de Snepscheut
___5143:TR:84	\$5.00	<i>General Interconnect Problem</i> , MS Thesis Ngai, John
___5140:TR:84	\$5.00	<i>Hierarchy of Graph Isomorphism Testing</i> , MS Thesis Chen, Wen-Chi
___5139:TR:84	\$4.00	<i>HEX: A Hierarchical Circuit Extractor</i> , MS Thesis Oyang, Yen-Jen
___5137:TR:84	\$7.00	<i>Dialogue Designing Dialogue System</i> , PhD Thesis Ho, Tai-Ping
___5136:TR:84	\$5.00	<i>Heterogeneous Data Base Access</i> , PhD Thesis Papachristidis, Alex
___5135:TR:84	\$7.00	<i>Toward Concurrent Arithmetic</i> , MS Thesis Chiang, Chao-Lin
___5134:TR:84	\$2.00	<i>Using Logic Programming for Compiling APL</i> , MS Thesis Derby, Howard
___5133:TR:84	\$13.00	<i>Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems</i> , PhD Thesis Lin, Tzu-mu
___5132:TR:84	\$10.00	<i>Switch Level Fault Simulation of MOS Digital Circuits</i> , MS Thesis Schuster, Mike
___5129:TR:84	\$5.00	<i>Design of the MOSAIC Processor</i> , MS Thesis Lutz, Chris

Caltech Computer Science Technical Reports

___5128:TM:84	\$3.00	<i>Linguistic Analysis of Natural Language Communication with Computers</i> , Thompson, Bozena H
___5125:TR:84	\$6.00	<i>Supermesh</i> , MS Thesis Su, Wen-king
___5123:TR:84	\$14.00	<i>Mossim Simulation Engine Architecture and Design</i> , Dally, Bill
___5122:TR:84	\$8.00	<i>Submicron Systems Architecture</i> , ARPA Semiannual Technical Report
___5114:TM:84	\$3.00	<i>ASK As Window to the World</i> , Thompson, Bozena, and Fred Thompson
___5112:TR:83	\$22.00	<i>Parallel Machines for Computer Graphics</i> , PhD Thesis Ulner, Michael
___5106:TM:83	\$1.00	<i>Ray Tracing Parametric Patches</i> , Kajiya, James T
___5104:TR:83	\$9.00	<i>Graph Model and the Embedding of MOS Circuits</i> , MS Thesis Ng, Tak-Kwong
___5094:TR:83	\$2.00	<i>Stochastic Estimation of Channel Routing Track Demand</i> , Ngai, John
___5092:TM:83	\$2.00	<i>Residue Arithmetic and VLSI</i> , Chiang, Chao-Lin and Lennart Johnsson
___5091:TR:83	\$2.00	<i>Race Detection in MOS Circuits by Ternary Simulation</i> , Bryant, Randal E
___5090:TR:83	\$9.00	<i>Space-Time Algorithms: Semantics and Methodology</i> , PhD Thesis Chen, Marina Chien-mei
___5089:TR:83	\$10.00	<i>Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits</i> , Lin, Tzu-Mu and Carver A Mead
___5086:TR:83	\$4.00	<i>VLSI Combinator Reduction Engine</i> , MS Thesis Athas, William C Jr
___5082:TR:83	\$10.00	<i>Hardware Support for Advanced Data Management Systems</i> , PhD Thesis Neches, Philip
___5081:TR:83	\$4.00	<i>RTsim - A Register Transfer Simulator</i> , MS Thesis Lam, Jimmy current supply only: see <i>Acta Informatica</i> 20, 301-313, (1983)
___5074:TR:83	\$10.00	<i>Robust Sentence Analysis and Habitability</i> , Trawick, David
___5073:TR:83	\$12.00	<i>Automated Performance Optimization of Custom Integrated Circuits</i> , PhD Thesis Trimberger, Steve
___5065:TR:82	\$3.00	<i>Switch Level Model and Simulator for MOS Digital Systems</i> , Bryant, Randal E
___5054:TM:82	\$3.00	<i>Introducing ASK, A Simple Knowledgeable System</i> , Conf on App'l Natural Language Processing Thompson, Bozena H and Frederick B Thompson
___5051:TM:82	\$2.00	<i>Knowledgeable Contexts for User Interaction</i> , Proc Nat'l Computer Conference Thompson, Bozena, Frederick B Thompson, and Tai-Ping Ho
___5035:TR:82	\$9.00	<i>Type Inference in a Declarationless, Object-Oriented Language</i> , MS Thesis Holstege, Eric
___5034:TR:82	\$12.00	<i>Hybrid Processing</i> , PhD Thesis Carroll, Chris
___5033:TR:82	\$4.00	<i>MOSSIM II: A Switch-Level Simulator for MOS LSI User's Manual</i> , Schuster, Mike, Randal Bryant and Doug Whiting
___5029:TM:82	\$4.00	<i>POOH User's Manual</i> , Whitney, Telle

# Caltech Computer Science Technical Reports

___5018:TM:82	\$2.00	<i>Filtering High Quality Text for Display on Raster Scan Devices,</i> Kajiya, Jim and Mike Ullner
___5017:TM:82	\$2.00	<i>Ray Tracing Parametric Patches,</i> Kajiya, Jim
___5015:TR:83	\$15.00	<i>VLSI Computational Structures Applied to Fingerprint Image Analysis,</i> Megdal, Barry
___5014:TR:82	\$15.00	<i>Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture,</i> PhD Thesis Lang, Charles R Jr
___5012:TM:82	\$2.00	<i>Switch-Level Modeling of MOS Digital Circuits,</i> Bryant, Randal
___5000:TR:82	\$6.00	<i>Self-Timed Chip Set for Multiprocessor Communication,</i> MS Thesis Whiting, Douglas
___4684:TR:82	\$3.00	<i>Characterization of Deadlock Free Resource Contentions,</i> Chen, Marina, Martin Rem, and Ronald Graham
___4655:TR:81	\$20.00	<i>Proc Second Caltech Conf on VLSI,</i> Seitz, Charles, ed.
___3760:TR:80	\$10.00	<i>Tree Machine: A Highly Concurrent Computing Environment,</i> PhD Thesis Browning, Sally
___3759:TR:80	\$10.00	<i>Homogeneous Machine,</i> PhD Thesis Locanthi, Bart
___3710:TR:80	\$10.00	<i>Understanding Hierarchical Design,</i> PhD Thesis Rowson, James
___3340:TR:79	\$26.00	<i>Proc. Caltech Conference on VLSI (1979),</i> Seitz, Charles, ed
___2276:TM:78	\$12.00	<i>Language Processor and a Sample Language,</i> Ayres, Ron



# Caltech Computer Science Technical Reports

Please PRINT your name, address and amount enclosed below:

name \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_ Country \_\_\_\_\_

Amount enclosed \$ \_\_\_\_\_

\_\_\_\_\_ Please check here if you wish to be included on our mailing list

\_\_\_\_\_ Please check here for any change of address

\_\_\_\_\_ Please check here if you would prefer to have future publications lists sent to your e-mail address.

E-mail address \_\_\_\_\_

Return this form to: Computer Science Library, 256-80, Caltech, Pasadena CA 91125

_____ 89-11	_____ 88-03	_____ 5223	_____ 5164	_____ 5089
_____ 89-10	_____ 88-02	_____ 5221	_____ 5160	_____ 5086
_____ 89-09	_____ 88-01	_____ 5220	_____ 5158	_____ 5082
_____ 89-07	_____ 5258	_____ 5215	_____ 5157	_____ 5081
_____ 89-06	_____ 5256	_____ 5214	_____ 5147	_____ 5074
_____ 89-05	_____ 5253	_____ 5212	_____ 5143	_____ 5073
_____ 89-04	_____ 5251	_____ 5210	_____ 5140	_____ 5065
_____ 89-03	_____ 5250	_____ 5207	_____ 5139	_____ 5054
_____ 89-02	_____ 5249	_____ 5205	_____ 5137	_____ 5051
_____ 89-01	_____ 5247	_____ 5204	_____ 5136	_____ 5035
_____ 88-22	_____ 5246	_____ 5202	_____ 5135	_____ 5034
_____ 88-21	_____ 5244	_____ 5200	_____ 5134	_____ 5033
_____ 88-20	_____ 5243	_____ 5198	_____ 5133	_____ 5029
_____ 88-19	_____ 5242	_____ 5197	_____ 5132	_____ 5018
_____ 88-18	_____ 5241	_____ 5195	_____ 5129	_____ 5017
_____ 88-17	_____ 5240	_____ 5194	_____ 5128	_____ 5015
_____ 88-16	_____ 5239	_____ 5193	_____ 5125	_____ 5014
_____ 88-15	_____ 5238	_____ 5190	_____ 5123	_____ 5012
_____ 88-14	_____ 5236	_____ 5189	_____ 5122	_____ 5000
_____ 88-13	_____ 5235	_____ 5185	_____ 5114	_____ 4684
_____ 88-12	_____ 5234	_____ 5184	_____ 5112	_____ 4655
_____ 88-11	_____ 5233	_____ 5179	_____ 5106	_____ 3760
_____ 88-10	_____ 5232	_____ 5178	_____ 5104	_____ 3759
_____ 88-07	_____ 5230	_____ 5174	_____ 5094	_____ 3710
_____ 88-06	_____ 5229	_____ 5172	_____ 5092	_____ 3340
_____ 88-05	_____ 5228	_____ 5168	_____ 5091	_____ 2276
_____ 88-04	_____ 5227	_____ 5165	_____ 5090	_____